Future Generation Computer Systems 105 (2020) 717–729

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Offloading decision methods for multiple users with structured tasks in edge computing for smart cities

Li Kuang [a], Tao Gong [a], Shuyin OuYang [a], Honghao Gao [b,*], Shuiguang Deng [c]

[a] School of Computer Science and Engineering, Central South University, Changsha, 410000, China
[b] Computing Center, Shanghai University, Shanghai 200444, PR China
[c] College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

ARTICLE INFO

Article history:
Received 9 September 2019
Received in revised form 15 December 2019
Accepted 25 December 2019
Available online 3 January 2020

Keywords:
Edge computing
Internet of things
Smart city
Offloading decision
Minimization

ABSTRACT

An edge computing system is an emergent architecture for providing computing, storage, control, and networking abilities, that is an important technology to realize Internet of Things and smart cities. In an edge computing environment, users can offload their computationally expensive tasks to offloading points, which may reduce the energy consumption or communication delay. There are a large number of offloading points and users in a system, and their tasks are structured. However, resources of offloading points are limited, and users have different preferences for energy consumption and communication delays. In this paper, we first establish a system model for the environment with multiple users, multiple offloading points, and structured tasks. Then, we formalize an offloading decision problem in such an environment as a cost-minimization problem, which is a NP-hard problem. Thus, we design a method based on backtracking to obtain its exact solution; the method's time complexity is, unfortunately, exponential with the number of offloading points. To reduce the complexity, a method based on an improved genetic algorithm and a method based on a greedy strategy are designed. Finally, we validate and compare three methods in terms of the total cost of all users, resource utilization of offloading points and execution time. The simulation results show that the last method performs the best.

1. Introduction

According to CISCO [1], the number of things connected to the Internet will be 50 billion by 2020, shaping a rich digital environment. Sensors, intelligent fixed and mobile platforms (e.g., smartphones, tablets and home gateways), massive-scale cloud infrastructures and other network-enabled devices will cooperate and interact to create value across many fields in smart cities [2], including intelligent transportation, intelligent medical, environmental monitoring, industrial control, aerospace, military reconnaissance, etc. In typical applications of IoT (Internet of Things), vast amounts of information are analyzed, released and calculated, and real-time, safe and reliable services should be provided.

Edge computing architectures have a natural advantage in supporting IoT applications, since they are a type of computing mode that is physically close to the data source; they integrate the core capabilities of network computing and storage applications to provide intelligent services nearby. It thus accelerates the upload and download of contents, services and applications across the network, giving consumers an uninterrupted, high-quality web experience [3,4]. In a word, edge computing can create a telecommunication service environment with high performance, low latency and high bandwidth.

The rapid development of IoT and mobile Internet has enabled enormous resource-sensitive applications (e.g., interactive games, augmented reality, face recognition, video surveillance and traffic monitoring) to run on IoT devices and users' devices [5–7]. However, due to the inherent limitations of the physical sizes of edge devices, they usually have only limited local resources, such as computing ability and battery energy [8,9]. To improve the performance and reduce local execution costs of edge devices, edge computing provide a key function to edge devices, namely, computational offloading, which is a technology that migrates computational-intensive tasks from edge devices to powerful server of offloading point to improve the performance and reduce the local execution costs of edge devices [10,11]. However, considering synthetically criteria such as the execution delay and energy consumption, computational offloading is not always beneficial for edge devices. Therefore, it is necessary to

* Corresponding author.
E-mail addresses: kuangli@csu.edu.cn (L. Kuang), GongTao17@csu.edu.cn (T. Gong), ouyang@csu.edu.cn (S. OuYang), gaohonghao@shu.edu.cn (H. Gao), dengsg@zju.edu.cn (S. Deng).

https://doi.org/10.1016/j.future.2019.12.039
0167-739X/

carefully compare the overhead and benefits when offloading a task. If offloading makes the edge device's benefit greater than its overhead, try to offload it; otherwise, do not offload it, that is, execute the task locally.

Edge devices (e.g., smartphones and tablet PCs) have become an essential part of people's daily lives because they can provide convenient communication services at any time and any location with the dramatic development of mobile information techniques [11–13]. In addition, as the requirements of users become increasingly complex and diverse, it is almost impossible for a single task to meet the requirement of users. Thus, multiple subtasks should be composed into a composite task with a certain structure to meet user requirements [14–16]. For example, in order to satisfy business traveling requirements, a structured-task that consists of three subtasks may be generated with a certain structure: weather forecast (t1), flight reservation (t2) and hotel reservation (t3). t2 and t3 can be executed in parallel, as they are independent of each other and only rely on the results provided by t1 [11]. This structural relationship should be considered when we make a computational offloading decision, since some subtasks can be done simultaneously if they can be executed in parallel, while some subtasks should be done sequentially if they should be executed in sequence with order dependency.

In this paper, first, we define an offloading decision problem in a specific scenario with multiple users, multiple offloading points and structured tasks through formal methods. Because past research works on offloading decision scenarios do not consider these three characteristics together, it is the most important difference from other research works. We propose a definition of components in a system model and mathematical descriptions for six offloading cases by analyzing structured tasks; these six offloading cases have not appeared in other research works, so it is worthwhile to study the scenarios proposed in this paper. Next, we formalize the offloading decision problem as a cost minimization problem under specific constraints. Third, we design several solutions based on three strategies, including backtracking, an improved genetic algorithm and a greedy algorithm. Finally, we conduct simulation experiments to get the optimal decisions of all users and compare their performances in terms of three criteria. The main contributions of this paper are summarized as follows.

- This paper presents a specific computational offloading model. It consists of multiple users and multiple offloading points, where the task of each user is structured and computational resources at each offloading point are limited. In current research work, there are few studies regarding offloading decision problems for a scenario with these features.
- We formalize a computational offloading decision problem as a cost minimization problem under constraints according to the characteristics of the concerned scenario. We design an offloading decision method based on backtracking to seek an exact solution. To reduce the time complexity, we design two more methods based on an improved genetic algorithm and a greedy strategy to find an approximate solution.
- We verify and compare the methods proposed in the paper. Three experiments are conducted, using the total cost of all users, resource utilization of all offloading points, and decision completion time as the criteria. We evaluate their performance as the numbers of users, offloading points and computational resources vary. The results show that the proposed methods are feasible, and the greedy-based strategy is the best among the three.

The rest of this paper is organized as follows. Section 2 analyzes the related work regarding offloading decision making. We present a system model for the specific scenario in Section 3. We then present three methods to solve the offloading decision problem in Section 4. We present their experimental results and analyze their performance in Section 5. Finally, we provide conclusions and future work in Section 6.

2. Related work

2.1. Main challenges

Edge computing is a new concept that has emerged in recent years that provides a wide range of flexible computational offloading services for edge devices with short execution delay and low energy consumption characteristics [1]. The research on computational offloading is mainly conducted to answer the following three questions [2,3]: (1) how to allocate edge cloud resources, i.e., how the edge server allocates its resources to minimize the execution costs of edge devices and how the edge server can perform load balancing when the tasks of edge devices are offloaded to the server [17,18]. The selection of offloading points can significantly influence not only the execution delay, as considered in [17] but also the power consumption of the offloading points. Hence, the main objective of [18] is to analyze the impact of the cluster size (i.e., the number of points performing computing) on both the execution latency of the offloaded application and the power consumption of the offloading points; (2) How to perform mobility management. i.e., how to ensure the continuity and stability of a user's services when users roam on an edge network and offload tasks to an edge server [19–23]. The related papers try to find an optimal decision policy regarding whether the VM migration should be initiated to minimize overall system cost (up to 32% and up to 50% reduction of average cost is achieved compared to the never- and always-migrate options, respectively [19]). Moreover, some papers aim to find a proper trade-off between VM migration cost and VM migration gain [20], minimizing execution delay [21], minimizing VM migration time [22], or maximizing overall throughput [23]. (3) How to make the optimal computational offloading decisions. Specifically, we determine whether tasks of user's devices should be offloaded to the edge server in order to decrease the local execution costs of a user's devices, and if tasks need to be offloaded, which parts of them should be offloaded and to where [11–13,21–29].

2.2. Offloading decision

MEC promises dramatic reduction in latency and mobile energy consumption, tackling the key challenges for materializing 5G vision. The promised gains of MEC have motivated extensive efforts in both academia and industry on developing the technology [30]. Research on computational offloading decisions has been an attractive and challenging area in the field of edge computing. Some researchers have studied the computational offloading decision problem in the scenario with a single user and single or multiple offloading points [11–13,24]. Deng et al. [24] divided user applications into interdependent atomic parts. They formalized the offloading problem as a 0–1 program, with 0 representing offloading and 1 representing no offloading, while the problem has a feasible solution at the exponential complexity, and the search for an optimal solution is highly complex. Deng et al. [11] studied calling multiple subservices in a service flow to meet their complex needs and decide whether to offload some subservices of the service flow. Liao et al. proposed an improved heuristic binary particle swarm optimization algorithm [27] to solve this problem, and the experimental results show that their method can reduce the energy consumption of user equipment by 25%. Liu et al. [12] found the optimal offloading decision through

an 1D search algorithm according to the application buffer queue status, available computing resources of edge devices and the server and the channel characteristics of the connection. Therefore, they minimized the execution delay in the result. Mao et al. [13] studied the dynamic voltage frequency spreading and energy harvesting techniques of edge devices. To minimize the edge device execution delay, a low-complexity Lyapunov optimized dynamic computing offloading algorithm is proposed. However, its drawback is that it does not take into account the energy consumption of the edge device because exhaustion of the battery hinders the network connection.

Some researchers study the problem of offloading decision in a multi-user and single-offloading-point scenario [9,31–43]. Shi et al. [9] extended the work in [11] from single-user to multi-user and proposed a User-Aware Offload Decision (MAGA) approach based on genetic algorithms to minimize the overall cost of the user delay and energy consumption. Chen et al. [31] study the edge computing multi-user computing offloading problem in a multichannel wireless interference environment. They find that the problem is NP-hard, and it is difficult to find the optimal solution through a centralized algorithm. They thus use a game theory method to make distributed and efficient computational offloading decisions. Deng et al. [32] study the trade-off between power consumption and transmission delay in edge cloud computing systems. They formulate a workload distribution problem. In the case of limited service delay, the minimum workload between an edge and the cloud is allocated, which is found by approximating the original problem to three subproblems of the corresponding subsystems. In [33], in order to minimize the global energy consumption of all users under delay constraints, Sardellitti et al. formalize an offloading decision problem as a joint optimization problem of allocating wireless communication resources and computing resources.

It can be seen from the above review that in the most scenarios, there is a single user and a single offloading point or multiple users and a single offloading point when designing an offloading model. Most of the existing studies assume that a user's task is a single one for simplicity. They consider that a user's transmission power and computing ability are fixed. The concerned scenario settings are relatively simple and do not conform to actual scenarios. In addition, in a single-user scenario, most of the work on computational offloading decisions assumes that a server has infinite resources. Therefore, in this paper, we first propose a scenario with multiple users and multiple offloading points. In this scenario, the task of each user is composed of multiple subtasks that are related to each other, and one user's structured task is not complete until all subtasks of the user's structured task are completed. In addition, we set the bounds of computational resources and communication resources in this scenario to better reflect actual edge computing conditions.

3. System model

3.1. Scenario composition description

As shown in Fig. 1, the concerned scenario includes multiple active areas, each of which is composed of multiple UDs (User's Devices) and multiple OPs (Offload Points). The scenario may be a school, a business building or a large mall. Multiple OPs are distributed according to user activity (this is not the scope of this study) in the entire scenario [44]. We assume that the connection between each pair of OPs is realized via a high-speed communication medium. The transmission rate is much faster than the transmission rate between UDs and OPs. The transmission delay of data to be processed between OPs can thus be considered negligible compared to the transmission delay between OPs and UDs.

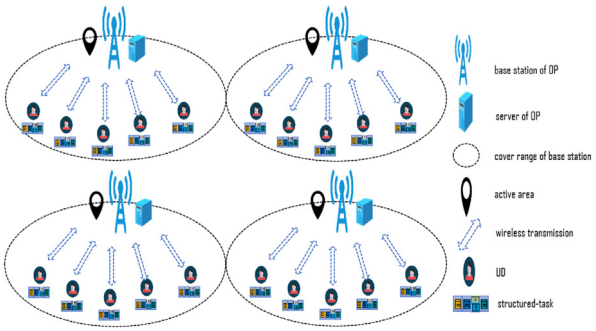

Fig. 1. Description of scenario.

The elements in our scenario are described as follows. A set of UDs is $M = \{UD_1, UD_2, \cdots, UD_M\}$, each UD has a structured task, and some subtasks of the structured-task may be computationally sensitive and may need to be migrated to OPs for execution. $N = \{OP_1, OP_2, \cdots, OP_N\}$ is a set of OPs, and the computational resources of each OP are limited. A UD can offload one of its subtasks to OP in an active area to which it is connected. When the computational resources of the OP to which a UD is connected to are not available, through the OP and high-speed communication medium, subtasks of the UD can be offloaded to other OPs whose computational resources are available (if offloading is still beneficial at that moment). The definitions and expressions of each element in our scenario are given as follows.

Definition 3.1 (*Structured Task*)**.** A structured task consists of a set of subtasks and a set of relations among subtasks. It is modeled as a 2-tuple $task = (T, R)$, where $T = \{t_1, \cdots, t_n\}$ is a set of subtasks, and $R = \{r(t_i, t_j)|t_i, t_j \in T\}$ defines their inter dependencies, such as independent, parallel and serial.

Definition 3.2 (*Subtask*)**.** According to [24],it is modeled as a 3-tuple $t_i = (d_i, d_o, w)$ where d_i and d_o are the sizes of the input data and output data, and w is the amount of CPU cycles that this subtask requires for execution.

Definition 3.3 (*UD*)**.** A UD refers to anybody or anything that need to offload its structured task. It is modeled as a 6-tuple $UD_m = (c_m, p_m^{exe}, p_m^{send}, p_m^{receive}, op_m, L_m)$, $m \in \{1, \cdots, M\}$, where c_m is the CPU speed (in MIPS) of UD_m, p_m^{exe} is the power consumption of UD_m when running tasks locally, and $p_m^{send}/p_m^{receive}$ is the power consumption of UD_m when sending and receiving data. Finally, op_m is the offloading point, i.e., the OP to which UD_m is connected, and L_m is the distance between the UD and OP.

Definition 3.4 (*OP*)**.** An OP is a place to which a UD can connect, which is a server with powerful computation ability. It is modeled as a 5-tuple $OP_n = (c_n, q_n^1, q_n^2, r_n^{up}, r_n^{down})$, $n \in \{1, \ldots, N\}$, where c_n is the CPU speed (in MIPS) of OP_n, q_n^1/q_n^2 is the amount of computational resources and communication resources, and $r_n^{\text{up}}/r_n^{\text{down}}$ is the upload rate and download rate,respectively, provided by the OP.

3.2. User cost description

We take a typical computation-sensitive task, i.e., face recognition, as an example to describe how a UD calculates the execution cost of its structured task. The dependency relationship of all subtasks of face recognition is shown in Fig. 2.

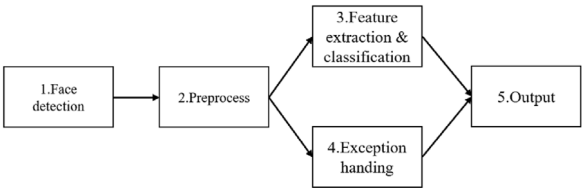

Fig. 2. The subtasks' dependency relations in face recognition.

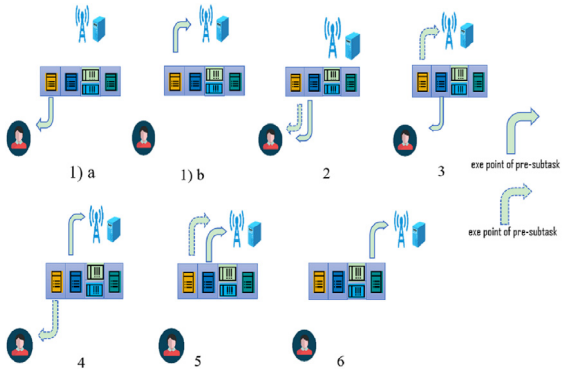

Fig. 3. Classification of execution point of subtask.

Definition 3.5 (*Delay Cost*)**.** This quantity is the consumed time that a structured task requires from start to end. Some subtasks of a structured task may execute at the UD or OP. It contains time consumption for execution and time consumption for communication.

Definition 3.6 (*Energy Consumption Cost*)**.** This quantity is the consumed energy of a UD because of executing a structured task. It contains energy consumption for local execution and energy consumption for communication.

As shown in each subgraph in Fig. 3, we classify the subtask execution locations into 6 cases. The execution cost of a subtask is described in terms of the UD's decision at the subtask execution point.

When calculating the delay cost, for two subtasks with parallel relation, since only when the two parallels subtask are completed can the subsequent subtask start, the greater of the two subtask delay costs should be taken as the delay for the process. The following is a delay cost expression for UD_m. g_m^k refers to the delay cost caused by the kth subtask of UD_m.

$$G_m = g_m^1 + g_m^2 + max(g_m^3, g_m^4) + g_m^5 \tag{1}$$

When calculating the energy consumption cost, the sum of the energy consumption generated by each subtask on UD is the energy consumption cost of the UD. The following is an expression of energy consumption cost of UD_m. e_m^k refers to the energy consumed by the kth subtask of UD_m.

$$E_m = e_m^1 + e_m^2 + e_m^3 + e_m^4 + e_m^5 \tag{2}$$

In the following expressions, t_m^{k-1} and t_m^k represent the (k-1)-th and the kth subtasks of UD_m, respectively. d_i^k and d_o^k is the size of the input data and output data of t_m^k, and d_o^{k-1} is the size of the output size of t_m^{k-1}. w is the amount of CPU cycles that t_m^k requires for execution. We assume that UD_m is connected OP_n; thus, c_m, p_m^{send}, $p_m^{receive}$, c_n, r_n^{up}, and r_n^{down} can be obtained from Definitions 3.3 and 3.4.

1. If current subtask t_m^k is the first subtask,

   (a) if t_m^k is executed at UD;

   $$g_m^k = \frac{w}{c_m}$$
   $$e_m^k = \frac{w}{c_m} * p_m^{exe}$$

   (b) if t_m^k is executed at OP.

   $$g_m^k = \frac{w}{c_n} + \frac{d_i^k}{r_n^{up}}$$
   $$e_m^k = \frac{d_i^k}{r_n^{up}} * p_m^{send}$$

2. If current subtask t_m^k is executed at UD, the previous one t_m^{k-1} is executed at UD.

   $$g_m^k = \frac{w}{c_m}$$
   $$e_m^k = \frac{w}{c_m} * p_m^{exe}$$

3. If current subtask t_m^k is executed at UD, the previous one t_m^{k-1} is executed at OP.

   $$g_m^k = \frac{w}{c_m} + \frac{d_o^{k-1}}{r_n^{down}}$$
   $$e_m^k = \frac{d_o^{k-1}}{r_n^{down}} * p_m^{receive}$$

4. If current subtask t_m^k is executed at OP, the previous one t_m^{k-1} is executed at UD.

   $$g_m^k = \frac{w}{c_n} + \frac{d_o^{k-1}}{r_n^{up}}$$
   $$e_m^k = \frac{d_o^{k-1}}{r_n^{up}} * p_m^{send}$$

5. If current subtask t_m^k is executed at OP, the previous one t_m^{k-1} is executed at OP.

   $$g_m^k = \frac{w}{c_n}$$
   $$e_m^k = 0$$

6. If current subtask t_m^k is a tail task and is executed at OP; finally, the result returns to UD.

   $$g_m^k = g_m^k + \frac{d_o^k}{r_n^{down}}$$
   $$e_m^k = e_m^k + \frac{d_o^k}{r_n^{down}} * p_m^{receive}$$

3.3. Formal description of the problem

From Section 3.2, we know that the total cost of one UD_m is

$$C(D_m) = \alpha_m * G_m + (1 - \alpha_m) * E_m \tag{3}$$

Therefore, the total cost of all UDs given by their cost preference is

$$F(D) = \sum_{m=1}^{M} \alpha_m * G_m + (1 - \alpha_m) * E_m \tag{4}$$

Our goal is to find the decision $D = \langle D_1, D_2, \ldots, D_M \rangle$ of all UDs that can minimize $F(D)$ while satisfying the resource constraints of all OPs. We know each UD's decision is $D_m = \langle d_m^1, d_m^2, \ldots, d_m^K \rangle$, $d_m^k \in N \bigcup \{0\}$, If $d_m^k \in N$, it means that the kth

subtask of UD_m is selected to be offloaded to an OP. If $d_m^k \in \{0\}$, it means that the kth subtask of UD_m is executed locally. $\alpha_m \in [0, 1]$ is UD's sensitivity of the experienced delay. When α_m is close to 0 , it indicates that UD is more sensitive to energy consumption; when α_m is close to 1 , it indicates that UD is more sensitive to experienced delay. The above are as follows:

$$\min \sum_{m=1}^{M} \alpha_m * G_m + (1 - \alpha_m) * E_m \tag{5}$$

$$subject\ to: \sum_{m=1}^{M} \sum_{k=1}^{K} sgn(d_m^k = n) \leq q_n^1\ for\ n \in N$$

sign() is a sign function. When the logical value in parentheses is true, the function value is 1; otherwise, it is 0. The constraint means that in all UDs' decisions, the number of all subtasks that are offloaded to one OP cannot exceed the number of computation resources of the OP, i.e., in a multiple-UD and multiple-OP scenario, we cannot exceed the computational resource limits.

4. Solutions

From [9,11,31], we can know (5) is an NP-hard problem even if M=1, and it is obviously more challenging to solve the offloading decision of multiple UDs. Therefore, the performance in terms of optimization results, decision efficiency and resource utilization is considered when we look for solutions of the problem. Therefore, we have designed the following three offloading decision methods based on backtracking, an improved genetic algorithm and a greedy strategy.

4.1. Offloading decision method based on backtracking

4.1.1. Backtracking

Since the problem that we aim to solve is NP-hard, and we want to acquire the minimum cost, that is, the exact solution of our problem, we can get the approximation ratio of another two algorithms proposed in Sections 4.2 and 4.3. Therefore, although the complexity of the backtracking-based algorithm is exponential with the number of offloading points, we propose it to get the exact solution.

Backtracking is a search method that searches forward according to the optimal conditions and then achieves the goal. It is a new evolutionary algorithm for solving optimization problems, and evolutionary algorithms are popular stochastic search algorithms that are widely used to solve nonlinear, nondifferentiable and complex numerical optimization problems [45]. However, when it explores a certain step and finds that the current choice is not excellent or fails to achieve the goal, it returns to the previous step and selects again. The idea of returning and reselecting is called backtracking, and the point that satisfies a backtracking condition is called a backtracking point. Many complicated, large-scale problems can be solved with backtracking. The general steps of solving problems using backtracking are as follows.

(a) Define the solution space of a given problem, which contains at least one optimal solution of the problem.

(b) Determine the solution space structure that is easy to search; it is easy to search the entire solution space with a backtracking method.

(c) Search the solution space via depth-first search or breadth-first search, and use a pruning function to avoid invalid searches during the search process.

4.1.2. Structured task offloading decision making

To solve the minimization problem given in Section 3.3, we first design an offloading decision method based on backtracking to get the exact solution. The steps of the structured task offloading decision method based on backtracking are introduced below.

(a) Define the solution space of our problem. We want to get the decision of all UDs. Each UD's decision is $\langle d_m^1, d_m^2, \ldots, d_m^K \rangle$, $d_m^k \in N \cup \{0\}$. It can be seen that the solution of this problem is an M*K-dimensional decision vector $D = \langle d^1, d^2, \ldots, d^{MK} \rangle$. In theory, the decision value of each dimension is $\{0, 1, 2, \ldots, N\}$ (included N OPs and UD). Therefore, the solution space of the problem is the discrete space $(N + 1)^{MK}$.

(b) Determine the structure of solution space that is easy to search. The search space of the problem solved by backtracking is mostly n-ary tree (n queen problem), subset tree (0–1 knapsack problem), and permutation tree (traveler problem). In this problem, because the constraint condition is that the computational resources do not exceed a certain limit, the decision value of each step can take any value of $\{0, 1, 2, \ldots, N\}$ in principle, so the solution space that can be constructed as an (N+1)-ary tree.

(c) Design a pruning function and search feasible solutions. In the process of searching, we design the pruning function based on the constraints of this problem and cut off the subtree without feasible solution, which can avoid invalid search and improve the search efficiency.

(d) Compare all feasible solutions and find the optimal solution from all feasible solutions. Through the search in step (c), all feasible solutions are substituted into the objective function F(D) to select the optimal solution.

4.1.3. Formal description

Symbol definition

k:the kth subtask

S^k:the set of value that kth subtask can take

d^k: the decision value of kth subtask

The pseudo code be shown in algorithm 1.

Algorithm 1: BTOD

Input : set of UDs : $M = \{1, 2, ..., M\}$
set of OPs : $N = \{1, 2, ..., N\}$
Output: Decision vector of all UDs : $< d^1, d^2, ..., d^{MK} >$

1 $k \leftarrow 1$
2 compute S^k
3 **while** $S^k \neq \emptyset$ **do**
4 $d^k \leftarrow min\{S^k\}$; $S^k \leftarrow S^k / \{d^k\}$
5 **if** $k < M * K$ **then**
6 $k \leftarrow k + 1$; $computeS^k$
7 **else**
8 $D =< d^1, d^2, ..., d^{KM} >$is a feasible solution
9 **end**
10 **if** $k > 1$ **then**
11 $k \leftarrow k - 1$; *goto* 3
12 **end**
13 **end**
14 return $argD\ min\ F(D)$

4.1.4. Complexity analysis and performance bound

The time complexity of backtracking in the worst case is the same as enumeration. The decision method based on backtracking makes a possible N+1 trial decision for each subtask of each UD. Therefore, the time complexity in the worst case is $T(M, N, K) = O((KM)^{N+1})$.

The essence of the backtracking algorithm is to enumerate all of the cases, find all feasible solutions, and then find the optimal solution in the feasible solution. Therefore, the search solution based on the backtracking algorithm's offloading decision must be the optimal solution under the given constraint conditions. However, please note that optimality here only refers to the cost, including the energy cost and delay cost. From the perspective of the time complexity of the algorithm, the algorithm is not practical because the time complexity increases exponentially with the number of offloading points.

4.2. Offloading decision method based on a genetic algorithm

4.2.1. Genetic algorithm

A genetic algorithm is a heuristic search algorithm that is often used in offloading decision for multiple users and structured tasks. However, besides the two constraints, we still have another constraint in our problem modeling, that is, multiple offloading points. Therefore, we propose to get the approximate solution by improving the cross, mutation operation and filtering individuals of the GA under condition of multiple offloading points.

A genetic algorithm is a method to search for optimal solutions by simulating natural evolutionary processes. Simulating a natural evolutionary process of human being results in stochastic optimization techniques called evolutionary algorithms (EAs) that can often outperform conventional optimization methods when applied to difficult real-world problems. Among them, genetic algorithms (GAs) are perhaps the most widely known types of EAs today [41]. A GA has the following steps.

(a) Coding. Encoding the solution of the problem, usually in binary and integer encoding.

(b) Initializing the population. Each individual in the initialized population is a possible solution. and the size of the population determine the size of the search space.

(c) Calculating individual fitness. To make genetic manipulation, the fitness of each individual in the population is calculated by the fitness function.

(d) Genetic manipulation. A selection operator chooses the same size good individuals from the previous generation; a crossover operator produces higher fitness individuals from the parents; a mutation operator makes the individual change toward higher fitness.

(e) Judging whether the rule of stop iteration is satisfied. When the preset maximum number of iterations is reached or the fitness of the best individuals in the continuous population no longer increases, the current result will be returned.

4.2.2. Structured task offloading decision making

To reduce the time complexity, we also use genetic algorithms to solve the problem. To adapt to the characteristics of this problem proposed in the paper, the genetic algorithm [46] is improved below. The following is the offloading decision method of solving one UD.

(a) Coding. For the computational offloading decision of a structured task, since the decision value can be 0 to N, we use integer encoding. An individual of the population in the genetic algorithm represents an offloading decision, which can be encoded as a K-dimensional vector, where K represents the number of subtasks in a structured task. In this coding scheme, each gene represents a subtask in a structured task, and the value of gene represents where the subtask is executed. Fig. 4 gives an example, where t^1 equals 0 refers to the first subtask executed at UD, and t^2 equals 3 refers to the second subtask executed at OP_3.

(b) Fitness function. The improved genetic algorithm uses function (3) in Section 3.3 as the fitness function to calculate the fitness value of each individual. The target value consists of two parts: the total execution time and energy consumption of a structured task.

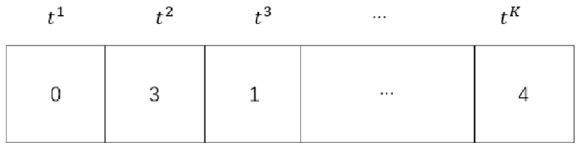

Fig. 4. Genetic encoding scheme.

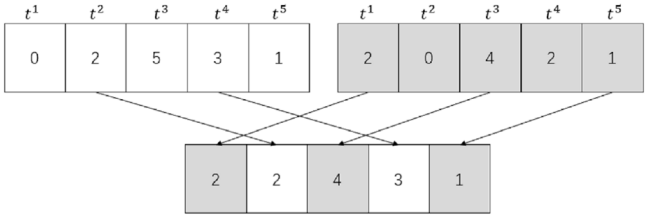

Fig. 5. Knowledge-based crossover.

(c) Genetic manipulation. (i) Initialization. In the initialization phase, the population size S, the maximum number of iterations I, the crossover probability p_c, and the mutation probability p_m are determined. $D_s = \langle d^1, d^2, \ldots, d^K \rangle$ is a UD's decision regarding each individual. d^k expresses whether the kth subtask of one UD should be offloaded. (ii) Selection. In this phase, individuals with high fitness are selected, and the next-generation population is generated via crossover and mutation operations. In this question, the individuals we choose are the ones with high fitness (small fitness value or small objective function value). We select the method based on roulette. The probability of selecting an individual for reorganization is related to its fitness value F_k. The method based on roulette is calculated as formula (6), where F_j is the fitness value of individual j. The selection strategy allows individuals with high fitness to have a greater likelihood of participating in the reorganization, which is the result that we want.

$$pr_k = 1 - \frac{F_k}{\sum_{j=1}^{S} F_j} \tag{6}$$

(iii) Crossover. A crossover operator combines the individuals from the selection phase, and it is wished to produce high quality offspring individuals. A standard single-point crossover operator is generally used but is often less effective when applied to specific problems. Therefore, we improve it to have a knowledge-based crossover according to the characteristics of the problem. We get only one individual from the parents D_1 and D_2 selected from the selection phrase, and each gene of the individual is a more adaptive one among its parents' counterpart genes. Compare the local fitness of each gene by calculating a weighted combination of execution time and energy consumption:

$$f_k = \alpha_m * g_m^k + (1 - \alpha_m) * e_m^k \tag{7}$$

g_m^k and e_m^k are the delay cost and energy cost of the kth subtask. A gene with a better local fitness will be selected as the gene of the offspring. As shown in Fig. 5. Bold genes mean they have higher local fitness. Therefore, the genes of offspring contain the t^2, t^4 from D_1 and the t^1, t^3, and t^5 from D_2. (iv) Mutation. Similarly, according to the characteristics of the problem, the mutation operator is improved similarly to the crossover operator. Calculate the probability of being mutated according to the fitness of each gene in the individual as shown in Fig. 6. Individuals from subtasks with higher execution delay and energy consumption

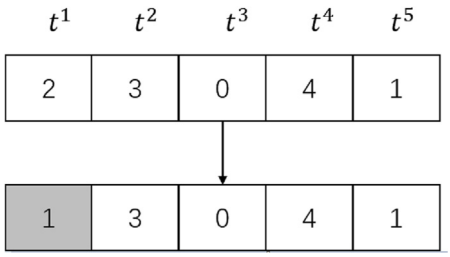

Fig. 6. Knowledge-based mutation.

will be more likely to mutate.

$$pr_k = 1 - \frac{f_k}{\sum_{i=1}^{K} f_i}, \tag{8}$$

(d) Judging whether a stopping iteration is satisfied. When the preset maximum number of iterations is reached or the fitness of the best individuals in the continuous population no longer increases, the current result will be returned.

In this decision problem, because there exist constraints with limited resources, there may exist new individuals who are not satisfied with the constraint after crossover manipulation or mutation manipulation and must be processed. Therefore, the genes that do not satisfy constraints will be changed to the best-fitness genes among offloading points with available or local resources. The offloading decision method based on an improved genetic algorithm (GAOD) randomly selects a user from the user set and performs the above process until all resources in offloading point are not available or each user has had an opportunity for offloading.

4.2.3. Formal description

Symbol definition.

S:the set of population

I:number of iterations

p_c, p_m: crossover probability and mutation probability

The pseudo code be shown in algorithm 2.

Algorithm 2: GAOD

Input : set of UDs : $\{1, 2, ..., M\}$
set of OPs : $\{1, 2, ..., N\}$
Output: Decision vector of all UDs : $< d^1, d^2, ..., d^{MK} >$

for *m=1 to M* **do**
  $D_m = \emptyset$
  initialize population $S = \{D_1, D_2, ..., D_S\}$, I, P_c, p_m
  repeat
    calculate the fitness $F(s)$of each individual in S
    initialize $newS = \emptyset$
    repeat
      select 2 individuals from according to (6)
      if *random*$(0, 1) < p_c$ **then**
        crossing according to (7)
      end
      if *random*$(0, 1) < p_m$ **then**
        mutation according to (8)
      end
      add offspring to the $newS$
    until ($|S|$ *offspring are created*);
    S=$newS$
  until (*optimal individual fitness not improving or number of iterations* $> I$);
  $D_m = arg_D \ min \ g_m(D_m)$
  update q_n^1 according to D_m
end

4.2.4. Complexity analysis and performance bound

The above offloading decision method is that all UDs simply invoke the improved genetic algorithm. From the above pseudocode analysis, the time complexity of the genetic algorithm depends on its population size. It is necessary to generate S individuals for each iteration, and one UD's decision needs to perform one iteration. The theoretical time complexity of the method is $O(ISM)$. We do not consider the complexity of producing an individual by crossover manipulation or mutation manipulation and the time for checking whether the new produced individual is satisfied with the constraint and handling new individuals who are not satisfied with the constraint. So, $O(ISM)$ is a lower bound on the complexity.

The algorithm is unfair to all users because in a limited-resources environment, the former selected users must first obtain computing resources, and subsequently, selected users will have no resources available.

4.3. Offloading decision method based on a greedy strategy

4.3.1. Greedy strategy

The backtracking-based algorithm and GA-based algorithm perform task offloading one user by one user; however, we think that it may also be important to guarantee fair opportunity of offloading decision for all users. Therefore, we propose a greedy-based algorithm, in which the subtasks of all users are layered, and the tasks in each layer perform offloading decisions according to a greedy strategy to guarantee fairness to users.

The greedy strategy is an improved hierarchical processing strategy. The design of the algorithm with a greedy strategy is carried out step by step. According to an optimization metric (which may or may not be objective function), the local optimal solution must be obtained at each step, and only one data point is considered in each step; besides, the selection by the greedy strategy should satisfy the local optimization conditions. This hierarchical processing method that can obtain an optimal solution in terms of a certain metric is called greedy strategy [47]. Although it cannot obtain an overall optimal solution for all problems, it can produce it for many problems (such as single-source shortest path and minimum spanning tree problem). In some cases, even if it cannot get the overall optimal solution, the final result can be near the optimal solution. It has the following steps.

(a) Establishing a mathematical model to describe the problem.

(b) Dividing the problem being solved into several subproblems.

(c) Solving each subproblem and getting the local optimal solution of the subproblem.

(d) Combining the local optimal solution of the subproblem into a solution of the original problem.

4.3.2. Structured-task offloading decision making

The backtracking-based method searches all subtasks decisions of one UD and then searches the next UD until all UDs' decisions are completed. The improved GA-based method gets a UD's decision by invoking the improved genetic algorithm and then solving the next UD until all UDs' decisions are completed. On the surface, they randomly select a UD from the set of UDs, and the offloading opportunity is fair to all UDs. However, we should note that the computational resources of our scenario in this paper are limited. It is impossible to satisfy the offloading

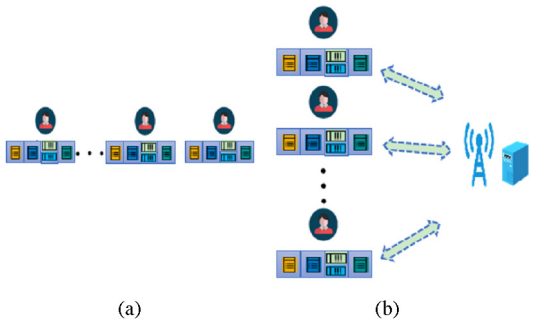

Fig. 7. Offloading forms of BTOD/GAOD(a) and GSOD(b).

requirements of all UDs (unless there are a few UDs only). In the case in which a large number of UDs exist in the offloading scenario, when the computational resources are exhausted, there must exist most UDs whose subtasks can be executed only locally, which is unfair to UDs that will be handled later. To make the offloading of the structured-task of all UDs more uniform, we design a greedy strategy-based method as follows:

(a) Establishing a mathematical model to describe our problem. According to the system model in 3.3., we can obtain a mathematical model of multiple UDs, multiple OPs, and structured task computational offloading decision problems, as specified in Eq. (5).

(b) Dividing the problem being solved into several subproblems. To make the offloading of structured tasks of all UDs more uniform, the method splits a UD structured task into several layers according to its structure. It should be noted that because a UD's task is structured, according to the UD cost model established in Section 3, only after a UD's kth subtask performs the offloading decision can the subsequent subtask of the kth subtask make an offloading decision. Therefore, we take the offloading decision problem of each layer's subtask as a subproblem. As shown in Fig. 7, the backtracking and improved GA-based methods are given in Fig. 7(a), while that of the greedy strategy-based one is given in Fig. 7(b).

(c) Solving each subproblem to obtain its optimal solution. The question is now how to make offloading decisions for all UDs' current kth subtasks to maximize the benefits of all current UDs. On this basis, we consider that the benefits of offloading structured tasks are related to the properties of the offloading environment and the properties of the structured tasks themselves. Therefore, we propose two greedy strategies from the perspective of an offloading environment and structured task in this paper. Greedy Strategy-(I) The difference between local execution cost and offloading execution cost. For the kth subtask of all UDs, the larger the difference in a subtask of a certain UD, the higher offloading priority the subtask has. Obviously, under the current conditions, the benefit of offloading the subtask of the UD is greater than the benefit of offloading any subtask of other UDs. Greedy Strategy-(II) The ratio of the amount of subtask calculation to the amount of data must be transmitted. For all kth subtasks of all UDs, the larger the ratio a subtask of a certain UD, the higher offloading priority the subtask is. Greedy strategy-I considers the benefit from the perspective of an offloading environment, while greedy strategy-II considers the benefit only from a structured task itself. Note that the ratio used in the latter exactly reflects the computational sensitivity corresponding to the concerned subtask.

(d) Combining the local optimal solution of the subproblem into a solution of the original problem. Through the third step, we get an M-dimensional decision vector of each layer, and each vector is a decision of all subtasks from M UDs. Then, the decisions of all the subtasks become the solution to the original problem.

Algorithm 3: GSOD

Input : set of UDs : {1, 2, ..., M}
set of OPs : {1, 2, ..., N}
Output: $D =< d_1^1, d_2^1, \ldots, d_M^1, d_1^2, d_2^2, \ldots, d_M^2, \ldots, d_1^K, d_2^K, \ldots, d_M^K >$

$k \leftarrow 1$
for $k \leftarrow 1$ *to* K **do**
  sort (UDs, strategy I or II)
  for $m \leftarrow 1$ *to* M **do**
    $d_m^k \leftarrow 0$
    $c_m^k \leftarrow \alpha_m * g_m^k + (1 - \alpha_m) * e_m^k$
    for $n \leftarrow 1$ *to* N **do**
      if $q_n^1 == 0$ **then**
        continue
      end
      $t\_d_m^k \leftarrow n$
      $t\_c_m^k \leftarrow \alpha_m * g_m^k + (1 - \alpha_m) * e_m^k$
      if $t\_c_m^k < c_m^k$ **then**
        $d_m^k \leftarrow t\_d_m^k$
        $c_m^k \leftarrow t\_c_m^k$
      end
    end
    if $d_m^k \neq 0$ **then**
      $q_n^1 \leftarrow q_n^1 - 1$
    end
  end
  $D[k][m] \leftarrow d_m^k$
end

4.3.3. Formal description

Symbol definition.
d_m^k:decision of kth subtask of UD_m
$D[k][m]$: the decision value indexed k*m in decision vector
The pseudo code is shown in algorithm 3.

4.3.4. Complexity analysis and performance bound

Time complexity analysis based on greedy strategy is relatively simple. In this method, the entire algorithm, from the first layer subtask to the Kth layer subtask, traverses M UDs, and N offloading points, from the beginning to the end of the subtask. All UDs are sorted according to the specified greedy strategy ($KlogM$) before making decisions on all of the UDs' kth subtasks. Therefore, the complexity of the algorithm is $O(MNK^2logM)$.

The greedy-based solution strategy we designed is to divide all same subtask of users into a layer. First, the first subtask of all users is sorted by the offloading gain, and we maintain a minimum heap for all subtask (the highest-benefit subtask has the highest priority). Then, when a user's subtask is completed, the algorithms will calculate offloading gain of the user's subsequent subtasks and push it into the heap. Therefore, this method is the optimal solution that is fairest to all users.

5. Experiments and results

In this section, we conduct four experiments. They evaluate the performance of offloading decision methods in terms of the total cost of all UDs, resource utilization of all OPs, and decision completion time of a certain method. The total cost of all UDs is the normalized sum of the delay cost and energy consumption cost of all UDs. The resource utilization of all OPs refers to the resources proportion occupied by UDs in all OPs. The decision completion time of a certain method is the consumed time that

Table 1
The parameters value of UDs and OPs.

Parameter	Value
Computation ability of UDs	$U[0.8, 1.8]$ (MIps)
Computation ability of OPs	$U[1.6, 2.6]$ (MIps)
Wireless transmission rate	$U[0.5, 0.9]$ (Mbps)
Execute power of UDs	$U[10, 20]$ (mj/s)
Send power of UDs	$U[1, 2]$ (mj/s)
Receive power of UDs	$U[2, 4]$ (mj/s)

Table 2
Parameter of the experiment 1.

Parameter	Setting1	Setting2	Setting3
Size of population	10–50	20	20
Number of iterations	600	100–600	600
Crossover probability	0.95	0.95	0.1–0.95

Table 3
Parameter of the experiment 2 to 4.

No of experiments	UDs	q_n^1	OPs
Experiment2	5–100	20	10
Experiment3	40	5–20	10
Experiment4	40	20	1–5

the method used in the process of finding the final offloading decision. For simple expression, we use c1, c2, and c3 to represent the total cost of all UDs, resource utilization of all OPs and decision completion time, respectively.

The experiments are intended to answer the following research questions.

(a) How does the number of UDs, number of OPs, and amount of computational resources affect the performance of the four offloading decision methods?

(b) For the offloading scenario in this paper, what is appropriate percentage of number of UDs to amount of computational resources?

(c) What is the performance of the four decision methods in the process of finding their optimal decisions?

5.1. Settings

In this paper, all experiments are run on a machine with an Intel Core i5 CPU at 2.5 GHz with 8 GB RAM, and all algorithms are implemented in Python 3.6. For convenience, the same structural sequence (such as face recognition application) is used for all UDs' structured tasks. Note that all the methods in this paper are applicable to different structural sequences. For each subtask, we generate the input/output data and workload based on a uniform distribution. In addition, we simulate the attribute parameters of UDs and OPs based on an actual situation. By following [48], we calculate the communication rate of OPs and UDs. For each experiment, each method is executed 200 times to draw convincing results.

Tables 1–3 report the experimental parameters. U in Table 1 means a uniform distribution.

5.2. Comparison of experimental results

The experiment is divided into four parts.

Experiment 1. Obtain appropriate GAOD parameters.

Experiment 2. Obtain relationship between number of UDs and evaluation criterion.

Experiment 3. Obtain relationship between total cost evaluation criterion.

Experiment 4. Obtain relationship between number of OPs and evaluation criterion.

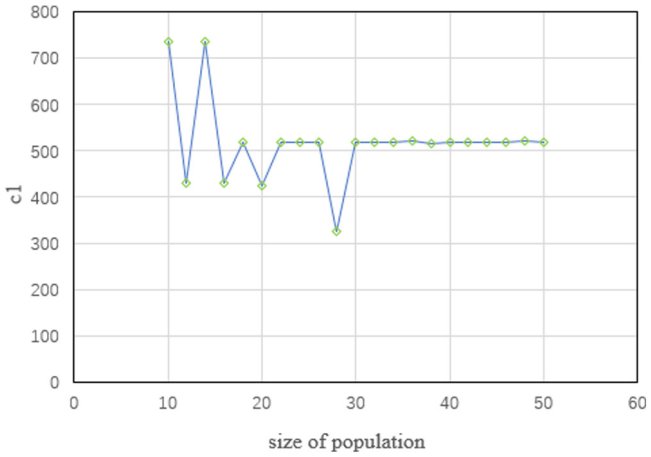

Fig. 8. Variation of c1 with size of population.

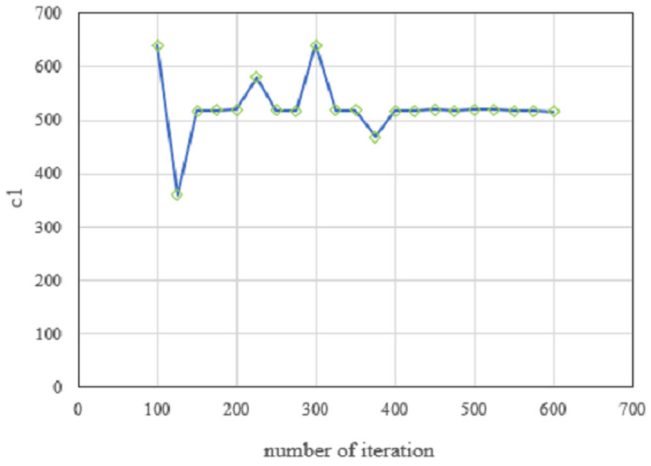

Fig. 9. Variation of c1 with number of iterations.

5.2.1. Obtaining the appropriate GAOD parameters

The performance of the genetic algorithm is related to the population size, number of iterations, and cross probability. Therefore, appropriate GAOD parameters should be set in comparison with the others. We need to do the following experiment. In this part, the number of UDs is 40, the number of resources in an OP is 20, and the number of OPs is 10. The following are the experimental results.

Fig. 8 shows that with increasing population size, the performance of GAOD improves, and because a large population means a large search range, more feasible solutions can be found. Fig. 9 shows that as the number of iterations increases, GAOD gradually becomes stable. Because the GA is an evolutionary algorithm, more iterations can produce better offspring. Fig. 10 shows the performance variation of GAOD with different crossover probability, which indicates that it is relatively stable with different crossover probability because the crossover probability only affects the rate of convergence. From the experimental results, we can observe that it is suitable when the size of population is 30 and the number of iterations is 325. Since the influence of crossover probability to GAOD is not particularly obvious, we set it to 0.8 to avoid slow convergence. More details about the GA can be found in [41].

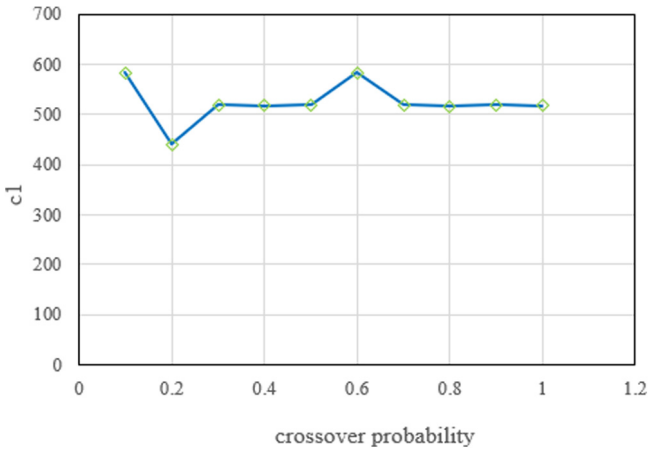

Fig. 10. Variation of c1 with crossover of probability.

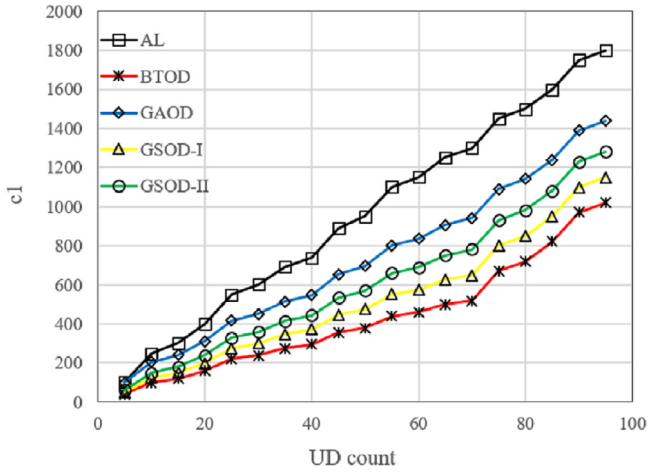

Fig. 11. Variation of c1 with UD count.

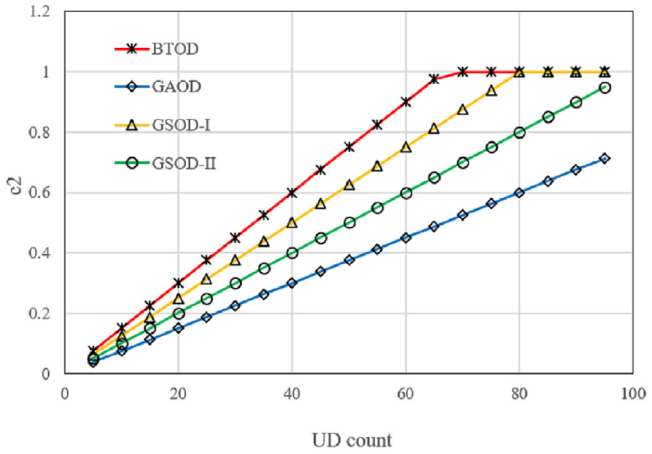

Fig. 12. Variation of c2 with UD count.

5.2.2. Obtaining the relationship between the number of UDs and evaluation criteria

To determine the relationship between four offloading decision methods and the number of UDs, we conduct the following experiments.

From Fig. 11, we can determine that BTOD is the best method for reducing the total cost of UDs, and GSOD-I is similar to BTOD, followed by GSOD-II and GAOD. BTOD gets the exact solution of the decision, and it is the best at reducing the total cost. GSOD-I and GSOD-II are based on a greedy strategy that does not guarantee getting the optimal solution but could get an approximately best solution. In addition, GSOD-I is better than GSOD-II at reducing total cost, which shows that the total cost of the UDs is not only related to the amount of workload and data needed to process structured tasks but also related to the offloading environment. The performance of GAOD is the worst because we need to deal with the situation that the computation resource constraint is not satisfied when we design GAOD, which takes a certain amount of time. Finally, the performances of the four methods are better than that of AL (always local execution). When the number of UDs increases to a certain threshold, approximately 70 UDs (as shown in Fig. 11), the increment of the total cost of UDs using four methods is approximately equal to that of AL. This is because some subsequent subtasks cannot get computational resources as the number of UDs increases. From Fig. 12, we can observe that with increasing UD count, resource utilization increases, too. BTOD has the highest resource utilization, followed by GSOD-I, GSOD-II and GAOD. Among them, GAOD has the lowest one. BTOD finds the most appropriate offloading point for each subtask to execute. Therefore, its resource utilization is the highest. As seen from Fig. 13, the decision completion time of BTOD grows fastest with the UD count, which is caused by exponential increase speed of the backtracking with the UD count. However, GSOD-I and GSOD-II show lower-order polynomial increase with UD count, while GAOD is in the middle. This is because GAOD takes extra time to deal with the conditions that do not satisfy computational resource constraints.

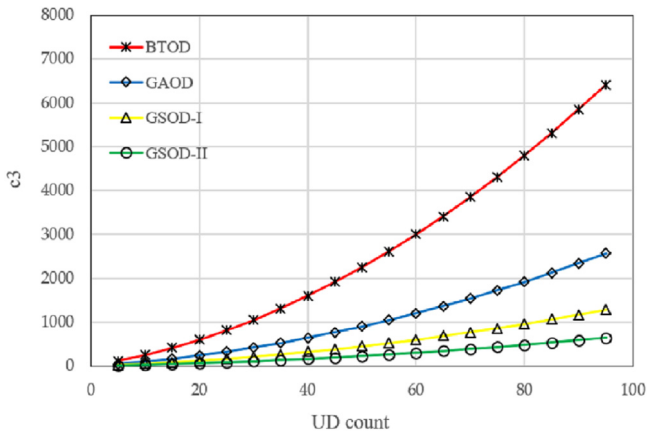

Fig. 13. Variation of c3 with UD count.

5.2.3. Obtaining the relationship between total number of resources and evaluation criteria

To determine the relationship between the four decision methods and total number of resources, we conduct the following experiments.

As can be observed from Fig. 14, with increasing number of resources in each OP, the total cost of the UDs decreases. However, when the count increases to a certain threshold, the cost decreases no longer because the computational resources may be sufficient for the current UDs, and increasing continuously the amount of resources cannot reduce the cost further. It can be determined from this experiment that an appropriate ratio of resource count to UD count is approximately 0.3. Fig. 15 shows that when the number of resources at an offloading point is small, the resource utilization is very high. At this time, if we continuously increase the number of resources, it maintains high

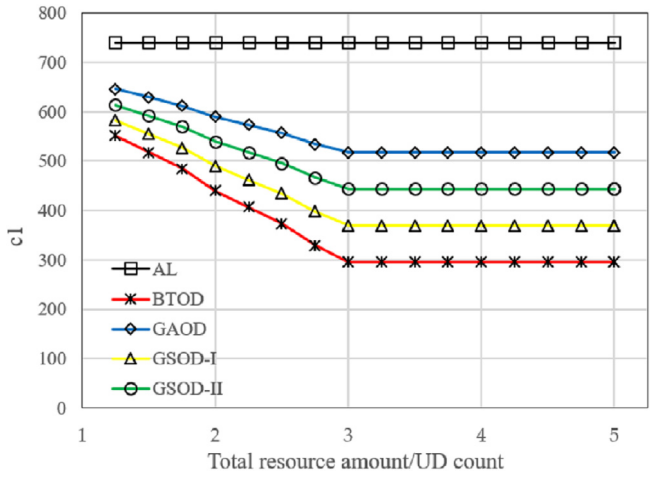

Fig. 14. Variation of c1 with resource amount/UD count.

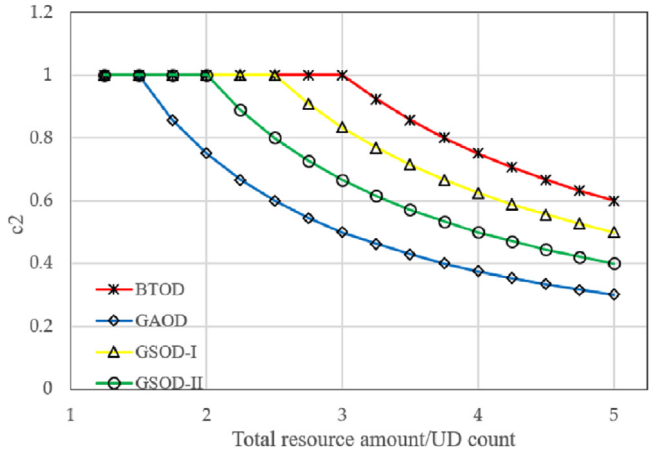

Fig. 15. Variation of c2 with resource amount/UD count.

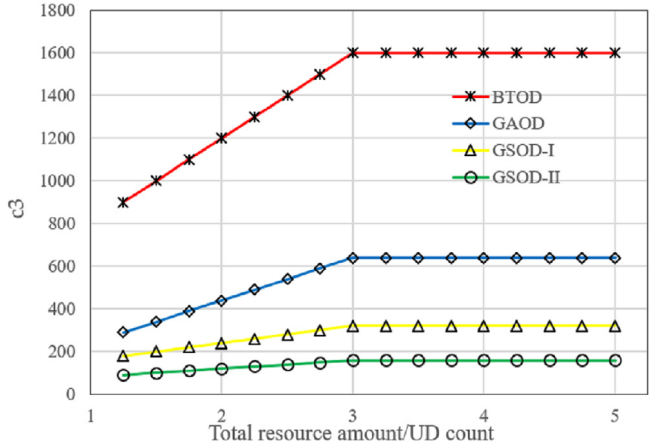

Fig. 16. Variation of c3 with resource amount/UD count.

utilization. However, it starts to decrease after a certain value is reached. This is because the resources may be sufficient for the current UDs, and increasing the amount of resources continuously leads to waste of resource. As can be observed from Fig. 16, when the number of resources increases, the decision completion time increases, because the four methods must judge whether the computational resources are available to make their own

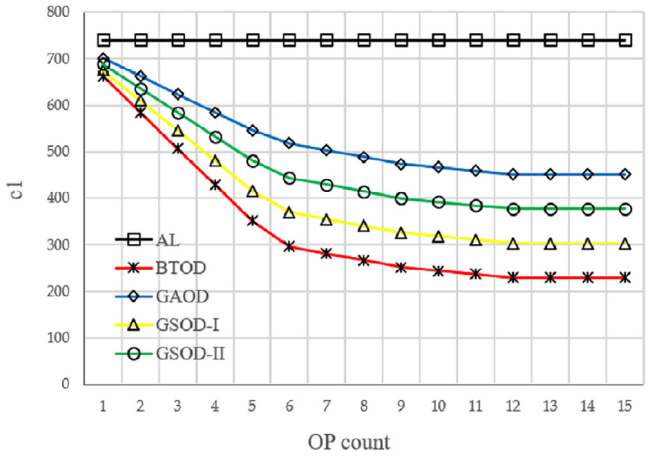

Fig. 17. Variation of c1 with OP count.

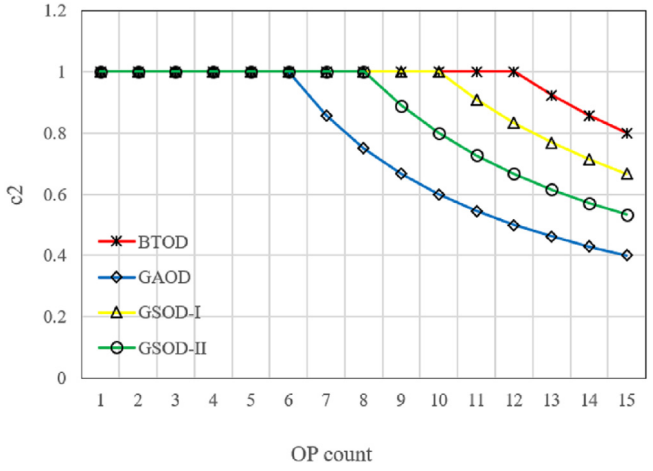

Fig. 18. Variation of c2 with OP count.

decisions when resources are insufficient. When a certain amount is increased, the decision completion time is not affected, because the execution time of the method is no longer affected by the number of resources at each OP when the number of resources exceed the needs of the UDs.

5.2.4. Obtaining the relationship between the number of OPs and evaluation criteria

To determine the relationship between the decision methods and the number of offloading points, we conduct the following experiment.

As can be observed from Fig. 17, with increasing number of OP, which means increasing number of resources, the total cost of the UDs decreases. However, when the number of OPs increases to a certain amount, the total cost of UDs does not decrease significantly. Because the current total number of resources can meet the offloading requirements of UDs, a continuous increase in the number of OPs has small impact on the criterion. We can learn from Fig. 18 that when the number of OPs is small, the resource utilization is very high. A continuous increase in the number of OPs maintains high utilization. When the number of OPs increases to a certain value, the utilization of computational resource will be reduced. The results are similar to those shown in Fig. 15, because the resources may be sufficient for the current UDs, and continuing to increase the number of OPs might lead to waste of resources. As can be observed from Fig. 19, with

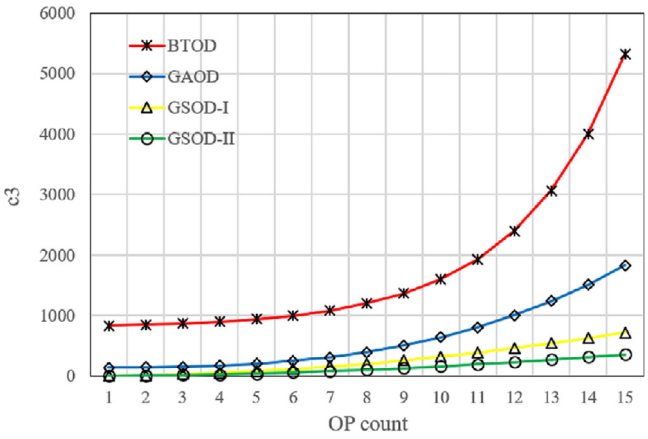

Fig. 19. Variation of c3 with OP count.

increasing number of OPs, BTOD is exponentially increasing because when a decision is made, BTOD has another try for each additional offloading point. GSOD-I and GSOD-II present a low-order polynomial level of growth, which is similar to the case of increasing the number of UDs. GAOD appears to exhibit a higher-polynomial-level growth than GSOD. This is because GAOD has a random decision value when increasing the number of OPs.

5.3. Summary of the experiments

From the above experiments, we can obtain the following conclusions: although BTOD is the best method in terms of optimizing the total cost of UDs, its exponential time complexity in terms of the number of OPs is unacceptable. For GSOD-I, although it is not as good as BTOD at optimizing the total cost of UDs, it has an approximate solution to BTOD's. An advantage of GSOD-I is that its time complexity is much lower than BTOD's. Regarding GSOD-II, it is slightly worse than GSOD-I at optimizing the total cost of UDs and slightly better in terms of time complexity than GSOD-I, but the difference is not obvious. GAOD is not as good as the above three methods in terms of total cost of UD optimization, and it is also worse than GSOD in terms of time complexity. Regardless of the method used, the resource utilization increases with the number of users. However, by comparison, it can be found that BTOD can reduce the average total cost of all UDs by approximately 60%, and GSOD-I can reduce the total cost of all UDs by approximately 55%. The decision completion time of BTOD is much longer than that of GSOD-I. Therefore, GSOD-I is more suitable for solving our concerned offloading decision problem.

6. Conclusion

In this paper, we study an offloading decisions problem in multiple-UD, multiple-OP, and structured-task scenarios. Specifically, we establish a system model that fits the scenario, formalize the problem as a total cost minimization problem, and then design four methods based on three strategies. Finally, we carry out four experiments and compare the proposed methods. The experiment results show that the method based on a greedy strategy is the best method by comparing three criteria used in this work; the method can reduce the total UD cost by approximately 55%. In this paper, we assume that the UD in the scenario is static in order to simplify the problem. In future work, we plan to consider a dynamic scenario. Machine learning methods [49,50] can be used to improve the performance.

Declaration of competing interest

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

CRediT authorship contribution statement

Li Kuang: Conceptualization, Methodology, Writing - review & editing, Funding acquisition, Supervision. **Tao Gong:** Investigation, Formal analysis, Writing- original draft. **Shuyin OuYang:** Software, Validation. **Honghao Gao:** Writing - review & editing, Supervision. **Shuiguang Deng:** Validation, Supervision.

Acknowledgments

This work was funded by National Natural Science Foundation of China (No. 61772560), National Key R&D Program of China (No. 2018YFB1003800).

References

[1] CISCO, The Internet of Things, Infographic, 2011. Available online at: http://blogs.cisco.com/news/the-internet-of-things-infographic.
[2] D. Nyriazis, T. Varvarigou, D. White, et al., Sustainable smart city IoT applications: heat and electricity management & eco-conscious cruise control for public transportation, in: 2013 IEEE 14th International Symposium on A World of Wireless, Mobile and Multimedia Networks, WoWMoM, IEEE, 2013, pp. 1–5.
[3] S. Barbarossa, S. Sardellitti, P.D. Lorenzo, Communicating while computing: distributed mobile cloud computing over 5g heterogeneous networks, IEEE Signal Process. Mag. 31 (6) (2014) 45–55.
[4] N. Fernando, S.W. Loke, W. Rahayu, Computing with nearby mobile devices: A work sharing algorithm for mobile edge-clouds, IEEE Trans. Cloud Comput. to be published. http://dx.doi.org/10.1109/TCC.2016.2560163.
[5] Y. Zhang, D. Niyato, P. Wang, Offloading in mobile cloudlet systems with intermittent connectivity, IEEE Trans. Mobile Comput. 14 (12) (2015) 2516–2529.
[6] Y.C. Liu, M.J. Lee, Y. Zheng, Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system, IEEE Trans. Mobile Comput. 15 (10) (2016) 2398–2410, http://dx.doi.org/10.1109/TMC.2015.2504091.
[7] G. Premsankar, M.D. Francesco, T. Taleb, Edge computing for the internet of things: a case study, IEEE Internet Things J. 5.2 (2018) 1275–1284.
[8] P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading, 2017, arXiv preprint arXiv:1702.05309.
[9] ETSI, Mobile edge computing (MEC): Technical requirements, V1.1.1, Mar. 2016.
[10] Y. Shi, S. Chen, X. Xu, Maga: a mobility-aware computation offloading decision for distributed mobile cloud computing, IEEE Internet Things J. 5.1 (2018) 164–174.
[11] S. Deng, L. Huang, J. Taheri, A.Y. Zomaya, Computation offloading for service workflow in mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (12) (2015) 3317–3329.
[12] J. Liu, Y. Mao, J. Zhang, K.B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, in: 2016 IEEE International Symposium on Information Theory, ISIT, IEEE, 2016.
[13] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. 34 (12) (2016) 3590–3605.
[14] Honghao Gao, Wanqiu Huang, Xiaoxian Yang, Applying probabilistic model checking to path planning in an intelligent transportation system using mobility trajectories and their statistical data, Intell. Autom. Soft Comput. 25 (3) (2019) 547–559.
[15] Honghao Gao, Wanqiu Huang, Yucong Duan, Xiaoxian Yang, Qiming Zou, Research on cost-driven services composition in an uncertain environment, J. Internet Technol. 20 (3) (2019) 755–769.
[16] B. Lin, W. Guo, G. Chen, N. Xiong, R. Li, Cost-driven scheduling for deadline-constrained workflow on multi-clouds, IPDPS Workshops 2015, pp. 1191–1198.
[17] M. Jia, J. Cao, W. Liang, Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks, IEEE Trans. Cloud Comput. 5 (4) (2017) 725–737.
[18] T. Zhao, S. Zhou, X. Guo, Y. Zhao, Z. Liu, A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing, in: 2015 IEEE Globecom Workshops (GC Wkshps), IEEE, 2015.
[19] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, K.K. Leung, Dynamic service placement for mobile micro-clouds with predicted future costs, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 1002–1016.

[20] X. Masip-Bruin, E. Marin-Tordera, G. Tashakor, A. Jukan, G.J. Ren, Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems, IEEE Wirel. Commun. 23 (5) (2016) 120–128.
[21] J. Song, Y. Cui, M. Li, J. Qiu, R. Buyya, Energy-traffic tradeoff cooperative offloading for mobile cloud computing, in: 2014 IEEE 22nd International Symposium of Quality of Service, IWQoS, IEEE, 2014.
[22] Y. Zhang, D. Niyato, P. Wang, Offloading in mobile cloudlet systems with intermittent connectivity, IEEE Trans. Mob. Comput. 14 (12) (2015) 2516–2529.
[23] N. Fernando, S.W. Loke, W. Rahayu, Computing with nearby mobile devices: a work sharing algorithm for mobile edge-clouds, IEEE Trans. Cloud Comput. (2016).
[24] M. Deng, H. Tian, B. Fan, Fine-granularity based application offloading policy in cloud-enhanced small cell networks, in: 2016 IEEE International Conference on Communications Workshops, ICC, IEEE, 2016.
[25] S. Mao, S. Leng, K. Yang, Q. Zhao, M. Liu, Energy efficiency and delay tradeoff in multi-user wireless powered mobile-edge computing systems, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017.
[26] S. Bi, Y.J. Zhang, Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading, IEEE Trans. Wireless Commun. 17 (6) (2018) 4177–4190.
[27] Z. Liao, L. Gao, T. Zhou, X. Fan, Y. Zhang, J. Wu, An oil painters recognition method based on cluster multiple kernel learning algorithm, IEEE Access (2019).
[28] X. Fan, Z. Chen, F. Cai, J. Wu, S. Liu, Z. Liao, Local core members aided community structure detection, Mobile Netw. Appl. 2 (2017) 1–9.
[29] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, in: 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 5, IEEE, 1997.
[30] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: the communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.
[31] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2016) 2795–2808.
[32] R. Deng, R. Lu, C. Lai, T.H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption, IEEE Internet Things J. 3 (6) (2016) 1171–1181.
[33] S. Sardellitti, G. Scutari, S. Barbarossa, Joint optimization of radio and computational resources for multicell mobile-edge computing, IEEE Trans. Signal Inf. Process. Netw. 1 (2) (2015) 89–103.
[34] C. Li, X. Zheng, Z. Yang, L. Kuang, Predicting short-term electricity demand by combining the advantages of arma and xgboost in fog computing environment, Wirel. Commun. Mobile Comput. 2018 (2018).
[35] L. Kuang, L. Yu, L. Huang, Y. Wang, P. Ma, C. Li, Y. Zhu, A personalized qos prediction approach for cps service recommendation based on reputation and location-aware collaborative filtering, Sensors 18 (5) (2018) 1556.
[36] S. Deng, L. Huang, J. Taheri, J. Yin, M. Zhou, A. Y, Zomaya:mobility-aware service composition in mobile communities, IEEE Trans. Syst. Man Cybern.: Syst. 47 (3) (2017) 555–568.
[37] L. Kuang, X. Yan, X. Tan, et al., Predicting taxi demand based on 3D convolutional neural network and multi-task learning, Remote Sens. 11 (11) (2019) 1265.
[38] K. Dou, B. Guo, L. Kuang, A privacy-preserving multimedia recommendation in the context of social network based on weighted noise injection, Multimedia Tools Appl. 78 (19) (2019) 26907–26926.
[39] Y. Zhao, G. Kou, Y. Peng, Understanding influence power of opinion leaders in e-commerce networks: an opinion dynamics theory perspective[j], Inform. Sci. 426 (2018) 131–147.
[40] R. Urena, G. Kou, Y. Dong, et al., A review on trust propagation and opinion dynamics in social networks and group decision making frameworks[j], Inform. Sci. 478 (2019) 461–475.
[41] M. Gen, L. Lin, Genetic Algorithms, in: Wiley Encyclopedia of Computer Science and Engineering., John Wiley Sons, Inc, 2008.
[42] J. Li, J. Wu, L. Chen, Block-secure: Blockchain based scheme for secure P2P cloud storage, Inform. Sci. 465 (2018) 219–231.
[43] S. Barra, M.De. Marsico, M. Nappi, et al., De marsico m nappi m others a hand-based biometric system in visible light for mobile environments[j], Inform. Sci. 479 (2019) 472–485.
[44] L. Chiaraviglio, F. Cuomo, A. Gigli, M. Maisto, Y. Zhou, Z. Zhao, H. Zhuang, A reality check of base station spatial distribution in mobile networks, in: 2016 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2016.
[45] Civicioglu, Pinar, Backtracking search optimization algorithm for numerical optimization problems, Appl. Math. Comput. 219 (15) (2013) 8121–8144.
[46] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A. Fast, Nsga-ii, IEEE Trans. Evolut. Comput. 6 (2) (2002) 182–197.
[47] D.Z. Du, K.I. Ko, X. Hu, Greedy Strategy. Design and Analysis of Approximation Algorithms, 2012.
[48] T.S. Rappaport, Wireless Communications: Principles and Practice, vol. 2, Prentice hall PTR, New Jersey, 1996.
[49] N. Zheng, K. Wang, W. Zhan, L. Deng, Targeting virus–host protein interactions: feature extraction and machine learning approaches, Curr. Drug Metab. (2018).
[50] L. Deng, J.C. Wang, J.P. Zhang, Predicting gene ontology function of human micrornas by integrating multiple networks, Front. Genet. 10 (2019) 3.

Li Kuang received her B.S. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 2004 and 2009 respectively. She is currently a full professor at the school of Computer Science and Engineering, Central South University.

Her research interests include Edge/Service/Mobile Computing. Up to now, and she has published more than 40 papers in journals and refereed conferences. She serves as a committee member of many international conferences, such as International Conference on Web Services, IEEE International Congress on Big Data, Asia-Pacific Services Computing Conference. She also serves as a reviewer of several international journals, such as IEEE Transactions on Services Computing, Signal Processing, Journal of Parallel and Distributed Computing, Frontiers of Computer Science and Information Sciences.

Tao Gong received the B.S. degree from the University of South China, Hengyang, China, in 2015, and he is currently pursuing the M.S. degree in software engineering at the School of Computer Science, Central South University, Changsha, China. His research interests include Edge Computing and Machine Learning.

Shuyin Ouyang is currently pursuing the B.S. degree in software engineering at the School of Computer Science, Central South University. His research interests include Edge Computing and Machine Learning.

Honghao Gao received the Ph.D. degree in Computer Science and started his academic career at Shanghai University in 2012. He is an IET Fellow, BCS Fellow, EAI Fellow, IEEE Senior Member, CCF Senior Member, and CAAI Senior Member. Prof. Gao is currently a Distinguished Professor with the Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, China. His research interests include service computing, model checking-based software verification, and sensors data application.

Shuiguang Deng is currently a full professor at the College of Computer Science and Technology in Zhejiang University, China, where he had received a BS and PhD both in Computer Science in 2002 and 2007, respectively. He previously worked at the Massachusetts Institute of Technology in 2014, and at Stanford University in 2015 as a visiting scholar.

His research interests include Edge/Service/Mobile Computing. He serves as Associate Editor for the journal IEEE Access and IET Cyber–Physical Systems: Theory & Applications as an Associate Editor. Up to now, he has published more than 100 papers in journals and refereed conferences. He is an IEEE Senior Member and was granted the Rising Star Award by IEEE TCSVC in 2018.