IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 11, 2019, accepted May 23, 2019, date of publication May 27, 2019, date of current version June 10, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2919319

An Energy-Efficient and Deadline-Aware Task Offloading Strategy Based on Channel Constraint for Mobile Cloud Workflows

YINGJIE WANG[1], LEI WU[1], XIUSHENG YUAN[1], XIAO LIU[2], AND XUEJUN LI[1], (Member, IEEE)

[1]School of Computer Science and Technology, Anhui University, Hefei 230601, China
[2]School of Information Technology, Deakin University, Geelong, VIC 3125, Australia

Corresponding authors: Lei Wu (wuleijsj@ahu.edu.cn) and Xiao Liu (xiao.liu@deakin.edu.au)

This work was supported in part by the Humanities and Social Sciences of MOE under Project 16YJCZH048 and in part by the Natural Science Foundation of Anhui Province under Project 1708085MF160.

ABSTRACT Energy efficiency is a fundamental problem due to the fact that numerous tasks are running on mobile devices with limited resources. Mobile cloud computing (MCC) technology can offload computation-intensive tasks from mobile devices onto powerful cloud servers, which can significantly reduce the energy consumption of mobile devices and thus enhance their capabilities. In MCC, mobile devices transmit data through the wireless channel. However, since the state of the channel is dynamic, offloading at a low transmission rate will result in the serious waste of time and energy, which further degrades the quality of service (QoS). To address this problem, this paper proposes an energy-efficient and deadline-aware task offloading strategy based on the channel constraint, with the goal of minimizing the energy consumption of mobile devices while satisfying the deadlines constraints of mobile cloud workflows. Specifically, we first formulate a task offloading decision model that combines the channel state with task attributes such as the workload and the size of the data transmission to determine whether the task needs to be offloaded or not. Afterward, we apply it to a new adaptive inertia weight-based particle swarm optimization (NAIWPSO) algorithm to create our channel constraint-based strategy (CC-NAIWPSO), which can obtain a near-optimal offloading plan that can consume less energy while meeting the deadlines. The experimental results show that our proposed task offloading strategy can outperform other strategies with respect to the energy consumption of mobile devices, the execution time of mobile cloud workflows, and the running time of algorithms.

INDEX TERMS Mobile cloud computing, task offloading, channel constraint, energy consumption.

I. INTRODUCTION

Mobile devices such as smart-phones and tablets have become an essential part of our daily lives due to their high convenience and efficiency. However, battery life is a critical factor affecting the Quality of Service (QoS) of mobile applications. In recent years, mobile cloud computing (MCC) [1] has become a promising way to enhance the capabilities of mobile devices. In MCC, task offloading technology [2] can migrate computation-intensive tasks to powerful servers in the cloud through a wireless network to reduce the energy consumption of mobile devices. However, task offloading will inevitably increase the communication between the mobile device and the cloud and thus increases the cost on the energy consumption, communication time and bandwidth [3], [4]. Therefore, the decision of task offloading is actually a decision on the tradeoff between the communication cost and the computation cost considering the QoS constrains of mobile workflow applications.

Many task offloading approaches have been proposed to find the optimal task offloading solutions for mobile applications running in MCC [5]–[7]. Li *et al.* [8] analyzed customer service preferences, especially when the details of service requests were not fully disclosed. Although it improved the transaction success rate and the user satisfaction rate, the energy consumption reduction of mobile devices was ignored. Kuang *et al.* [9] proposed a strategy which can coordinate task offloading between mobile devices and produce

The associate editor coordinating the review of this manuscript and approving it for publication was Parul Garg.

offloading solutions with less response time to meet the deadlines. However, they focused only on single applications rather than different types of mobile workflow applications. Moreover, these existing approaches cannot perform efficiently when there are dynamic changes in the channel state. If the data transmission rate is too slow during task offloading, channel congestion may occur, which could significantly increase the execution time and the energy consumption. The channel state directly affects the data transmission rate. For example, if a task with large transmission data but a small workload is offloaded when the channel state is bad, it will produce much larger time overhead and energy consumption compared with running the task locally on the mobile device. Therefore, it is essential to find an effective task offloading strategy which can simultaneously consider the impact of the channel state and reduce the energy consumption of the mobile device for different mobile workflows.

In this paper, we propose an energy-efficient and deadline-aware task offloading strategy based on the channel constraint for mobile cloud workflows. Specifically, we first formulate a task offloading decision model in which the channel state and task attributes such as the workload and the size of transmission data are combined to determine whether the task should be offloaded to the cloud server or not. Afterwards, we propose a new channel constraint based adaptive inertia weight based particle swarm optimization (CC-NAIWPSO) strategy to find the near-optimal offloading decision, which can prevent the premature convergence of the particle swarm optimization (PSO) algorithm and significantly reduce the energy consumption of the mobile device. The experimental results demonstrate the better performance of CC-NAIWPSO in reducing the energy consumption of the mobile devices, satisfying the deadlines of different types of workflows and reducing the algorithm running time.

The remainder of this paper is organized as follows. Section II presents the related work. Section III illustrates an example of task offloading that motivates this work. Section IV defines all the models that are used in this paper and Section V provides the novel CC-NAIWPSO strategy in detail. Section VI demonstrates the experimental results. Finally, Section VII concludes the paper and provides some potential future work

II. RELATED WORK

A. MOBILE CLOUD COMPUTING

In the last decade, many studies have been dedicated to the research and application of cloud computing [10]–[13], which is a new distributed computing paradigm that can provide users with on-demand and reliable services. Recently, with the rapid development of the Internet of things (IoT), new computing paradigms such as MCC [14]–[16], Fog Computing (FC) [17]–[19] and Mobile Edge Computing (MEC) [20]–[22] have been proposed according to the requirements of different application scenarios. In this paper, since our focus is on mobile workflow applications, MCC is selected as the computing paradigm. Zhang *et al.* [23] developed an energy efficient approach to jointly optimize task offloading and radio resource allocation for lower energy consumption in MEC. It can offload tasks to the MEC servers on the edge of the network that is closer to the users. Wei and Jiang [24] proposed an algorithm for FC to solve the nonconvex online optimization problem with polynomial complexity using the Lyapunov optimization method, which can improve the offloading efficiency and maximize the long-term average system utility. Since FC and MEC is still in its infancy, many more works have been devoted to the study of task offloading in MCC, which aims to reduce the execution time of mobile applications and the energy consumption of mobile devices for the purpose of satisfying user's requirements.

B. ENERGY CONSUMPTION REDUCTION

With the emergence of a large number of mobile applications, how to extend the mobile battery lifetime has become a hot research topic. Considering the heterogeneity of multiple remote sites, Zhang *et al.* [25] combined a local optimization algorithm with genetic search, which can reduce the energy consumption for the mobile device as much as 90% to 98.5%. Chen *et al.* [26] established an offloading algorithm model by using the population based incremental learning (PBIL) algorithm to achieve load balance of servers when offloading large-scale tasks. While these approaches are efficient in reducing the energy consumption, they did not consider time deadlines. However, deadlines cannot be overlooked as if the execution time exceeds the deadline, the user's satisfaction rate will be significantly deteriorated. Lin *et al.* [27] determined the frequencies for executing local tasks by applying the dynamic voltage and frequency scaling technique, which can reduce the energy consumption of the mobile devices in order to satisfy the deadlines. Li *et al.* [28] extended the work in [27] to select the appropriate voltage level for each task and adopted an energy-efficient scheduling algorithm called QCWES for sorting tasks in the workflow so as to meet the deadlines and reduce the energy consumption at the same time.

C. INTELLIGENT ALGORITHMS FOR TASK OFFLOADING

Computation offloading is an NP-hard problem [29] which can be addressed by intelligent algorithms. Popular intelligent algorithms include Tabu Search, Genetic Algorithm (GA), Ant Colony Optimization (ACO), PSO [30]–[34], and there are some works try to combine different intelligent algorithms to achieve even better performance [35]–[37]. Different intelligent algorithms have different advantages. Xu and Zhu [38] structured an adaptive penalty function based on GA to offload tasks within deadlines. The algorithm can adjust the population crossover and mutation probability to effectively avoid premature convergence of the algorithm and improve its reliability, and hence to reduce the energy consumption for task execution. GA has a good effect in solving small-scale tasks offloading, but for medium-scale or large-scale tasks

offloading, ACO algorithm usually can search for the optimal solution with fewer iterations and thus have better performance than GA [39]. He and Bai [40] developed a method based on the improved ACO to find the offloading solution which can improve the pheromone update and the optimal path selection so as to effectively accelerate the convergence rate and reduce the energy consumption. However, the method can be easily trapped into the local optimum.

In this paper, we first present a new adaptive inertia weight-based particle swarm optimization (NAIWPSO) algorithm that can constantly and more correctly adjust the particle velocity with time and energy constraints. In this way, it can significantly avoid the local optimum and help stabilize the convergence rate of the algorithm more. Afterwards, we design a task offloading decision model with the channel constraint and applied it to the NAIWPSO algorithm to form the CC-NAIWPSO strategy, which can intelligently adjust the offloading plans intelligently according to different channel states to produce the approximate optimal solution for the energy savings problem of mobile cloud workflows with deadline requirements. Experimental results show that our strategy can significantly reduce the energy consumption of mobile devices while satisfying the deadlines of mobile cloud workflows.

III. MOTIVATING EXAMPLE

In this section, we use an example workflow to demonstrate an instance of task offloading in detail. The motivating example shows that as the channel state changes, it is necessary to find a task offloading plan with lower energy consumption before the deadline expired.

Fig. 1 illustrates a workflow model with six tasks. Assume that the clock frequencies of the small, medium and large virtual machines are 1000 MHz, 1300 MHz and 1600 MHz, and the clock frequency of the mobile device is 500 MHz. In addition, suppose that when the channel state is good, the data transmission rate is 100 Kb/s. The workload of tasks is shown in Table 1, and the input data and output data of tasks are shown in Table 2 [41]. Tasks A and F are respectively the entrance and exit tasks of workflow that must be executed on the mobile device, and thus there is no transmission data for task A and task F.

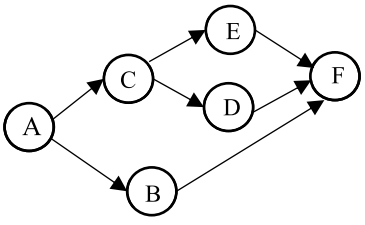
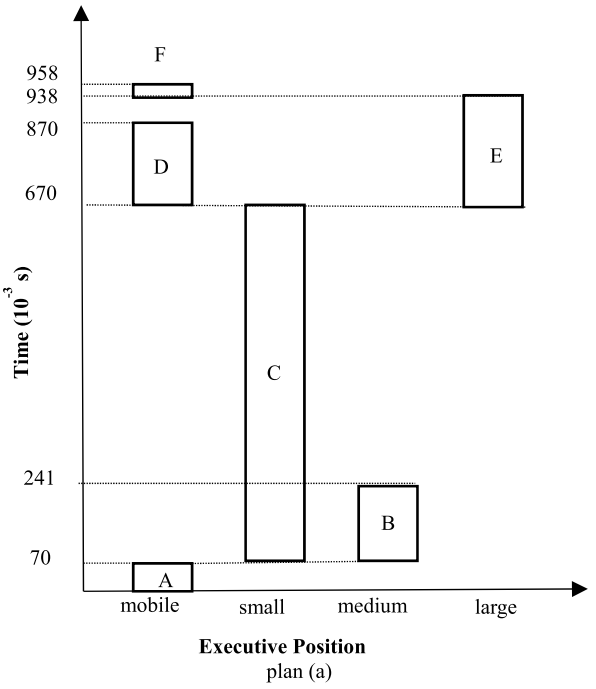


FIGURE 1. Example of workflow.

Supposing that the deadline of the workflow in Fig. 1 is 1 s and the channel state is good, two offloading plans that satisfy the deadline are given in Fig. 2. Plan (a) randomly offloads tasks whereas plan (b) offloads tasks considering the impact of the channel state. Apparently, the final execution



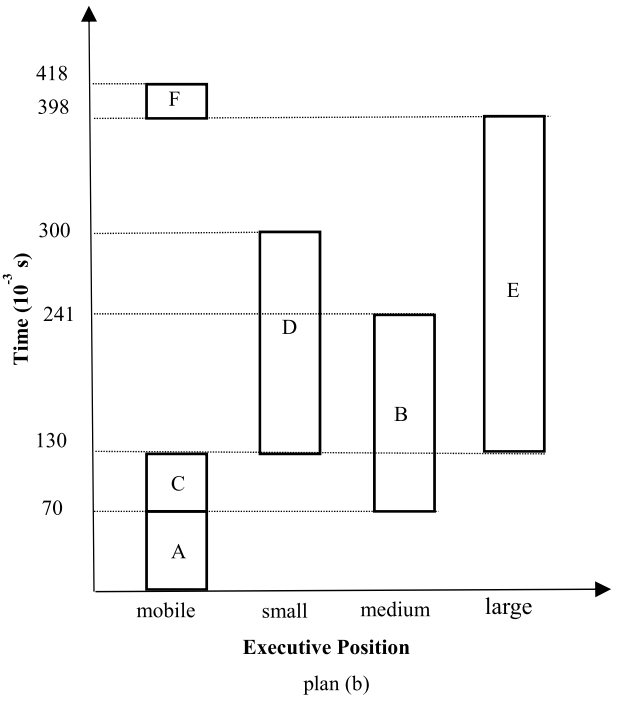


FIGURE 2. Gantt charts of two offloading plans (R = 100 Kb/s).

time of these two plans both meet the cut-off time constraint that is mentioned above. For the former, the total execution energy is 0.214 J. For the latter, the total energy consumption is 0.108 J, which is 0.106 J (approximately 50%) lower than the former. Moreover, the execution time of the channel constrained offloading plan decreases by 0.540 s.

As shown in table 3, suppose that the channel state turns bad at this moment and the data transmission rate drops to 20 Kb/s. At this point, the previous plan b is set as plan b* in

TABLE 1. Workload of tasks.

	A	B	C	D	E	F
Workload (M cycle)	35	40	30	100	60	10

TABLE 2. Input and output data of tasks.

	B	C	D	E
Input Data (Kb)	10	25	4	8
Output Data (Kb)	4	32	3	15

TABLE 3. Offloading plans in different channel states (deadline = 1 s).

Channel State	Offloading Plans	Execution Time (s)	Energy (J)
R=100 Kb/s	plan (a)	0.958	0.214
	plan (b)	0.418	0.108
R=20 Kb/s	plan (b)*	**1.338**	0.240
	plan (c)	0.563	0.203

the current state, the energy consumption rises to 0.240 J and the execution time rises to 1.338 s. Apparently, the execution time is now over the deadline of 1 s. Due to the change in channel state, we desperately need a decision that is constrained by the channel state to form a new plan (c), as shown in Fig. 3, which transforms tasks B and E to the mobile device from the cloud servers and moves D from the small to large virtual machine. Hence, the adjusted plan (c) satisfies the deadline and its energy consumption reduces to 0.203 J.

From the analysis above, it can be seen that when the channel state is good, task D possesses 100 Kb of the workload but only 4 Kb and 3 Kb of the input and output data, respectively. Therefore, task D should be preferentially executed on the cloud. Nevertheless, the input and output data of task C reach up to 25 Kb and 32 Kb, respectively, whereas the workload of task C is merely 30 Kb, which means that task C prefers to be executed on the mobile device. After we leverage an offloading decision to restrict task C and task D, task D is offloaded to the small virtual machine and task C is executed on the mobile device such that the workflow execution time and energy consumption are significantly reduced. When the channel state becomes bad, it is no longer appropriate for data transmission. Therefore, some tasks that are performed on the virtual machines need to be transferred to the mobile device to avoid unnecessary energy consumption.

In conclusion, when the channel state is unstable, we urgently need an effective task offloading strategy with the channel constraint decision to find the offloading plans with lower energy consumption within the deadline.

IV. SYSTEM MODEL

In this section, we will define all the models that are used in our paper in detail. First, we introduce the model for the workflow and MCC system. Then, we present the execution time model and the energy consumption model that are used for task offloading. Finally, the channel model is introduced.

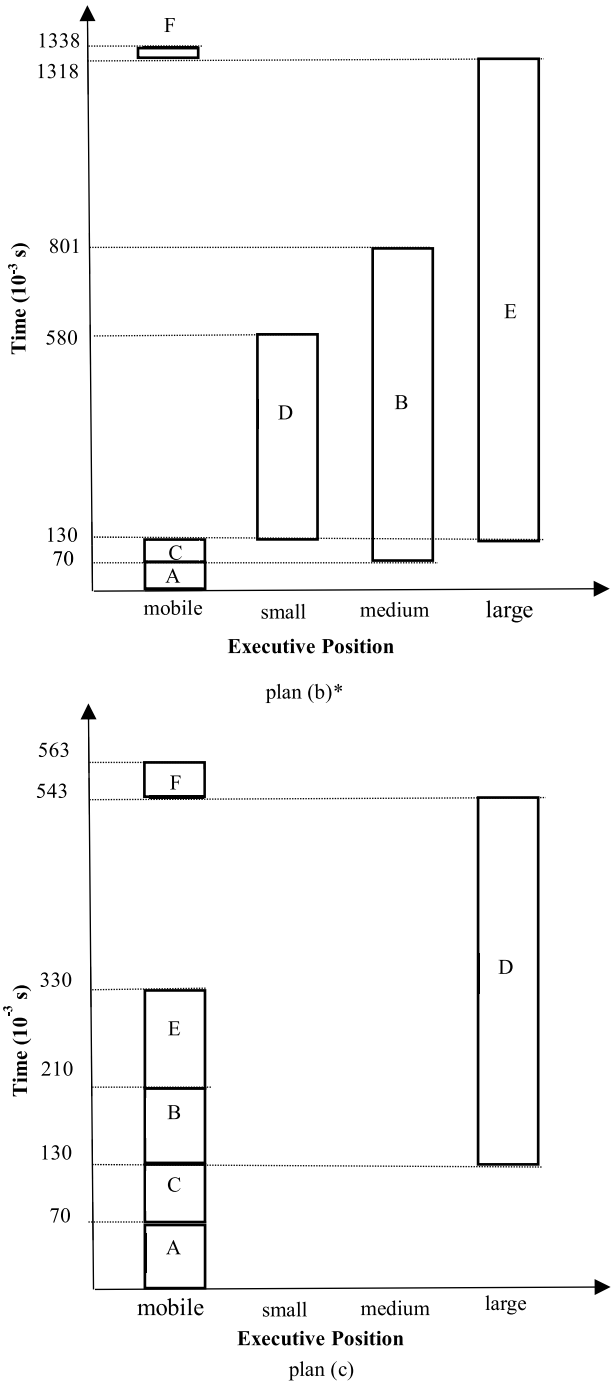


FIGURE 3. Gantt charts of two offloading plans (R = 20 Kb/s).

A. WORKFLOW MODEL

In a workflow, we use a Directed Acyclic Graph (DAG) to represent the dependency relationship between tasks [27]. $G = (V, E, w)$ is a directed acyclic graph with n nodes. The set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of the ordered executable tasks. The directed edge $e_{ij} = (v_i, v_j) \in E$ illustrates the invoked relationship between tasks v_i and v_j in which task v_j starts executing only when task v_i has finished. Furthermore, the set of node weights $w = \{w_1, w_2, \ldots, w_n\}$ describes the computational workload or the amount of CPU

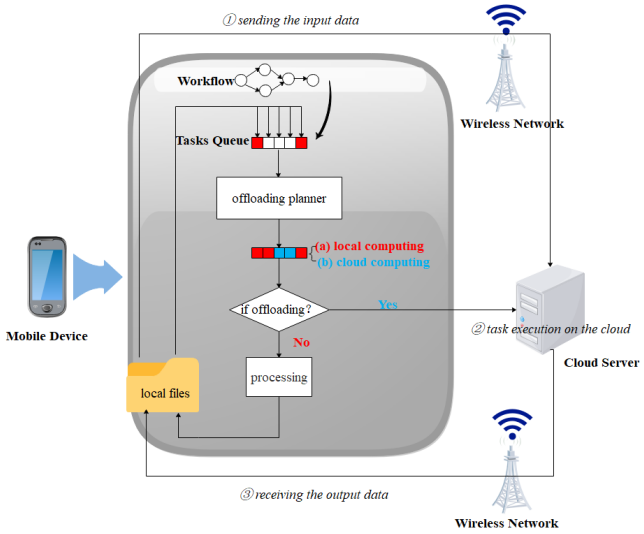


FIGURE 4. The framework of MCC system model.

cycles (in million cycles, M cycles) for each task. If task v_i is offloaded to the cloud server for execution, the mobile device needs to transmit the input data to the cloud, which is denoted as $inputdata_i$. Similarly, when task v_i is executed on the cloud, the mobile device will receive the output data from the cloud, which is denoted as $outputdata_i$. In addition, there must be an entrance node without any predecessors and an exit node without any successors in this graph. The entrance node corresponds to the entrance task in the workflow and the exit node corresponds to the exit task. They must be executed on the mobile device [42].

B. MCC SYSTEM MODEL

Fig. 4 depicts the framework of our proposed MCC system model. In the gray box, the process of task offloading is described. Here, we use variable x_i to denote the offloading plan of task v_i. Specifically, $x_i = 0$ denotes that the task will be executed on the mobile device and $x_i = 1$ denotes that the task will be offloaded to cloud servers. The task offloading plans can be expressed as the sets of $X = \{x_1, x_2, \ldots, x_n\}$. Since the entrance and exit tasks must be executed on the mobile device, x_0 and x_{n+1} are always equal to 0, which can also be shown in Fig. 4.

In Fig. 4, offloading planner is the main component in our framework and is actually a decision maker. It decides what task can be run locally and what task can be offloaded. Offloading planner can produce the optimal offloading plan by executing certain task offloading strategy. In this paper, we aim to propose an optimal task offloading strategy to minimize energy consumption under the deadlines of mobile workflows.

As shown in Fig. 4, a task can be either executed locally on the mobile device or remotely on the cloud server according to its offloading plan. If task v_i will be offloaded to the cloud server, there are three phases in sequence: ① sending the input data, ② task execution on the cloud, ③ receiving the output data. In the first phase, the $inputdata_i$ is sent to the cloud server by the mobile device through the wireless network channel. In the second phase, task v_i is executed by the cloud server. In the third phase, the mobile receives the $outputdata_i$ from the cloud server to check the validity of the result before executing the next task.

When a task is offloaded to the cloud, we use R to represent the data transmission rate of the wireless network channel. The power when the mobile device is being idle is P_0 and the clock frequency of cloud server is f_c, where both P_0 and f_c are constants that can be measured. P_s and P_r denote the power for sending and receiving data of the mobile device, where P_s is considerably larger than P_r [27].

C. EXECUTION TIME MODEL

The workflow execution time contains two parts: the computation time and the communication time. As depicted in the MCC system model, if a task is executed on the cloud, it is necessary to send the input data to the cloud before execution and send the output data back to the mobile device after the task has finished. Therefore, if a task is executed on the mobile device, only the computation time is included. Meanwhile, if the task is executed on the cloud, besides the computation time, the communication time is also included. We formulate the execution time of task v_i as follows. All the relevant definitions for the symbols in the formulas below are shown in table 4.

1) COMPUTATION TIME

a: EXECUTED LOCALLY

If task v_i is executed on the mobile device ($x_i = 0$), then the computation time of task v_i can be calculated using (1).

b: EXECUTED REMOTELY

If task v_i is executed on the cloud ($x_i = 1$), then the computation time of task v_i can be calculated also using (1).

$$T_i^{comp}(x_i) = (1 - x_i) \times \frac{w_i}{f_m} + x_i \times \frac{w_i}{f_c} \tag{1}$$

2) COMMUNICATION TIME

a: EXECUTED LOCALLY

If tasks are executed on the mobile device ($x_i = 0$), there will not be any communication time that is generated.

b: EXECUTED REMOTELY

If task v_i is executed on the cloud ($x_i = 1$), then the communication time of task v_i can be calculated using (2).

$$T_i^{comm}(x_i) = x_i \times \frac{inputdata_i + outputdata_i}{R} \tag{2}$$

In addition, to obtain the execution time, we must calculate the start time of each task using (3).

$$T_{start}(x_i) = \begin{cases} T_{start}(x_0) = 0, & i = 0 \\ max_{x_j \in P(x_i)} \left\{ T_{start}(x_j) + T_j^{comp}(x_j) + T_j^{comm}(x_j) \right\}, & i \neq 0 \end{cases} \tag{3}$$

TABLE 4. Notations.

Notation	Definition
w_i	The computational workload for task v_i
x_i	The offloading plan of task v_i
P_m	The power of the mobile device when executing tasks locally
P_0	Idle power of the mobile device
f_m	The clock frequency of the mobile device
f_c	The clock frequency of the cloud server (different cloud servers have different f_c values)
R	The data transmission rate
P_s	The power of the mobile device when sending data
P_r	The power of the mobile device when receiving data
$inputdata_i$	The input data of v_i transformed from the mobile device to the cloud
$outputdata_i$	The output data of v_i transformed from the cloud to the mobile devices.
P(x_i)	the corresponding offloading plans of all the precursor tasks set of v_i

Thus, we can obtain the workflow execution time using (4), where T_{n+1}^{finish} is the finish time of the exit task v_{n+1}.

$$T(X) = T_{n+1}^{finish} = T_{start}(x_{n+1}) + T_{n+1}^{comp}(x_{n+1}) \quad (4)$$

D. ENERGY CONSUMPTION MODEL

When the tasks of the workflow are executed, the energy consumption of the mobile device also contains two parts: the computation energy consumption and the communication energy consumption. Note that in this work, our focus is to reduce the energy consumption of the mobile device through task offloading technology, and hence the energy consumption of cloud servers is not considered. As depicted in the MCC system model, if a task is executed on the cloud, it is necessary to send the input data to the cloud before execution and send the output data back to the mobile device after the task has finished. Therefore, a task that is executed on the mobile device only involves the computation energy consumption. When being executed on the cloud, the task also includes the energy consumption of sending and receiving the data in addition to the computation energy consumption. Thus, the energy consumption of task v_i can be formulated as follows.

1) ENERGY CONSUMPTION FOR COMPUTATION

a: EXECUTED LOCALLY

If task v_i is executed on the mobile device ($x_i = 0$), then the computation energy consumption of task v_i can be calculated using (5).

b: EXECUTED REMOTELY

If task v_i is executed on the cloud ($x_i = 1$), then the computation energy consumption of task v_i can be denoted as (5).

$$E_i^{comp}(x_i) = (1 - x_i) \times P_m \times \frac{w_i}{f_m} + x_i \times P_0 \times \frac{w_i}{f_c} \quad (5)$$

2) ENERGY CONSUMPTION FOR SENDING INPUT DATA

a: EXECUTED LOCALLY

If tasks are executed on the mobile device ($x_i = 0$), there will be no sending energy consumption.

b: EXECUTED REMOTELY

The energy consumption of task v_i for sending input data can be calculated using (6) if task v_i is executed on the cloud ($x_i = 1$).

$$|E_i^s(x_i) = x_i \times P_s \times \frac{inputdata_i}{R} \quad (6)$$

3) ENERGY CONSUMPTION FOR RECEIVING OUTPUT DATA

a: EXECUTED LOCALLY

If tasks are executed on the mobile device ($x_i = 0$), there will be no receiving energy consumption.

b: EXECUTED REMOTELY

The energy consumption for receiving output data can be calculated using (7) if task v_i is executed on the cloud ($x_i = 1$).

$$E_i^r(x_i) = x_i \times P_r \times \frac{outputdata_i}{R} \quad (7)$$

Therefore, we can obtain the energy consumption of mobile devices when the tasks are executed on the mobile device using (8) and on the cloud using (9)

$$E_m = \sum_{i=0}^{n} E_i^{comp}(x_i) \quad (8)$$

$$E_c = \sum_{i=0}^{n} \left[E_i^{comp}(x_i) + E_i^s(x_i) + E_i^r(x_i)\right] \quad (9)$$

In conclusion, the total energy consumption of mobile devices can be denoted as (10).

$$E(X) = E_m + E_c \tag{10}$$

E. CHANNEL MODEL

Mobile devices transmit data to the cloud through the wireless channel. Suppose that the data transmission rate is fixed and the wireless channel determines the transmission quality. Then, we can fit the wireless channel according to the Gilbert-Elliott (GE) channel model [43].

It is assumed that the channel state g_t can be detected in the time interval t. When the current state of the channel is good, then we have $g_t = g_G$. Similarly, when the state of the channel is bad, we have $g_t = g_B$. Suppose that the power of the mobile device when sending and receiving data is constant, which means that the channel state g_t is uniquely determined by the data transmission rate R_t in the time interval t. In addition, when $g_t = g_G$, we have $R_t = R_G$, where R_G is the data transmission rate when the state of the channel is good. When $g_t = g_B$, we have $R_t = R_B$, where R_B is data transmission rate when the channel state is bad. Furthermore, we can suppose that P_{BG} is the probability when the channel state changes from bad g_B to good g_G and P_{GB} is the probability when the channel state changes from good g_G to bad g_B. Thus, we can obtain that the stability probability of the channel state being good is $\frac{P_{BG}}{P_{BG}+P_{GB}}$ and the stability probability of the channel state being bad is $\frac{P_{GB}}{P_{BG}+P_{GB}}$.

In conclusion, the channel expectation transmission rate can be derived using (11).

$$Exp(R) = \frac{P_{BG}}{P_{BG}+P_{GB}}R_G + \frac{P_{GB}}{P_{BG}+P_{GB}}R_B \tag{11}$$

V. THE PROPOSED CC-NAIWPSO STRATEGY

In this section, we first introduce the task offloading decision model with the channel constraint. This decision model will be used in conjunction with the NAIWPSO algorithm to construct our task offloading strategy. Then, we present the definition of the fitness function. The fitness function is a crucial part of the NAIWPSO algorithm that aims to minimize the energy consumption of mobile devices given the deadline. Finally, CC-NAIWPSO is presented, which can solve the energy-saving problem of mobile devices while satisfying the deadlines of mobile workflow applications.

A. OFFLOADING DECISION MODEL WITH CHANNEL CONSTRAINT

In the process of offloading the task, since the state of the wireless channel is not fixed, it will generate more time and energy consumption when the data transmission rate is low. In addition, the workload and transmission data of the task will also affect the offloading plan.

Therefore, we formulate an offloading decision model based on the channel state and the attributes of the task to determine whether the task needs to be offloaded to the cloud server. The task offloading plan is further constrained in terms of the execution time and energy consumption of the mobile devices. In this way, the offloading plan can continuously approach the best solution. The offloading decision consists of the following two scenarios.

1) If task v_i is determined to be executed on the mobile device but

$$\frac{inputdata_i + outputdata_i}{w_i} < \frac{Exp(R)}{f_m}, \tag{12}$$

then instead of being executed on the mobile device, the task will be offloaded to the cloud for execution. We can take advantage of this condition to constrain the task in terms of the execution time.

2) If task v_i is determined to be executed on the cloud but

$$\frac{inputdata_i \times P_s + outputdata_i \times P_r}{w_i} \geq \frac{Exp(R) \times P_m}{f_m}, \tag{13}$$

then task v_i will be executed on the mobile device rather than being offloaded to the cloud server. Likewise, this condition can constrain the task for the purpose of reducing the energy consumption of the mobile device.

To summarize, the offloading decision that is based on the channel constraint can modify the offloading plan using formulas (12) and (13), which is very effective at dramatically reducing the execution time and energy consumption of the mobile device.

B. NAIWPSO ALGORITHM

The PSO algorithm has the characteristics of fewer parameters, easy implementation and fast convergence, and has been widely used in various optimization fields. However, the PSO algorithm does not effectively control the velocity of particles, and so it cannot well balance the global and local search abilities of particles. It also has the defect of local convergence, which results in the inability to find the optimal solution in a limited number of iterations. Therefore, we put forward the NAIWPSO algorithm. By comparing the fitness of each particle with the global optimal value, the state of particles can be described more accurately, thus improving the adaptability of the weight. The new weight can adjust the particle speed more accurately, and so that the algorithm can better balance the global and local search abilities of the particles and avoid falling into the local optimum. Furthermore, we can apply the algorithm to solve the problem of task offloading in MCC.

1) NEW ADAPTIVE INERTIA WEIGHTS (NAIW)

First, we calculate the success value of a single particle, as shown in equation (14).

$$S(i,t) = \begin{cases} 1, & fitness\left(pbest_i^t\right) < fitness\left(pbest_i^{t-1}\right) and \\ & fitness\left(pbest_i^t\right) < globalbest^t; \\ 0.5, & fitness\left(pbest_i^t\right) < fitness\left(pbest_i^{t-1}\right) and \\ & fitness\left(pbest_i^t\right) \geq globalbest^t; \\ 0, & fitness\left(pbest_i^t\right) = fitness\left(pbest_i^{t-1}\right); \end{cases} \tag{14}$$

where $S(i,t)$ denotes the success value of particle i at the t-th iteration, and $pbest_i^t$ is the optimal position of particle i at the t-th iteration. $fitness\left(pbest_i^t\right)$ is the fitness value of particle i in the optimal position at the t-th iteration, and $globalbest^t$ is the global optimal value at the t-th iteration.

Then, we calculate the success rate of the entire particle swarm using (15).

$$P_s(t) = \sum_{i=1}^{n} S(i,t)/n \tag{15}$$

where $P_s(t)$ represents the success rate of the particle swarm at the t-th iteration, $\sum_{i=1}^{n} S(i,t)$ is the sum of the success values of all particles, and n is the number of particles in the entire particle swarm.

Finally, the success rate is used to update the inertia weight using formula (16). The weight NAIW in the NAIWPSO algorithm will change with the different search states of the particle swarm, and therefore the particle velocity can be changed dynamically, as shown in formula (17).

$$\begin{aligned} \omega(t) &= (\omega_{max} - \omega_{min})P_s(t) + \omega_{min}, \\ & 0 \le \omega_{min} \le \omega_{max} \le 1 \end{aligned} \tag{16}$$

$$v_i(t) = \omega(\mathrm{t})\, v_i(t-1) + r_1 c_1\left(p_i^t - x_i^t\right) + r_2 c_2\left(p_g^t - x_i^t\right) \tag{17}$$

where $\omega(\mathrm{t})$ represents the inertia weight at the t-th iteration that is used to adjust the initial velocity of the particle at each iteration; $v_i(t)$ is the velocity of particle i at the t-th iteration;r_1 and r_2 denote random numbers between 0 and 1; c_1 and c_2 are learning factors; p_i^t and p_g^t are the individual extremum of the particle and the global extremum of the population, respectively; and x_i^t represents the position of particle i at the t-th iteration.

If the particle swarm is close to the best solution, the value of $P_s(t)$ will be relatively low. Accordingly, the value of $\omega(t)$ and the velocity of each particle will decrease for the purpose of refining the optimal solution at a lower search speed. In contrast, $P_s(t)$ will be relatively high if the particle swam is away from the optimal solution. Similarly, the value of $\omega(t)$ and the velocity of each particle will increase so that the particle swam can approach the optimal solution faster.

To sum up, the NAIWPSO algorithm can constantly change the velocity of particles according to the current position of the particle swarm through the NAIW to dynamically balance and improve the global search and local search abilities of the algorithm.

2) FITNESS FUNCTION

The fitness value can measure the quality of the individuals in the population. The objective of this strategy is to minimize the energy consumption of the offloading plan within the deadline, and so the fitness function is denoted as (18). If the execution time T(X) is less than or equal to the deadline, then $f_1 = 1$ and $f_2 = 0$. Otherwise, we set $f_1 = 0$ and $f_2 = 1$.

$$fitness = f_1 E(X) + 10 f_2 \times \frac{T(X)}{deadine} E(X) \tag{18}$$

Strategy: Energy-Efficient Task Offloading Strategy With Channel Constraint (CC-NAIWPSO)

Input: the particles N, the iterations, the tasks, the virtual machine VM_s and the deadline

Output: the offloading plan X_{best}

For i = 1 to N **do**
  Generate the initial task offloading plans and its search velocity randomly
End for
For i = 1 to N **do**
  Modify the offloading plans using the offloading decisions of (12) and (13)
  Calculate the fitness value of each plan using (18)
End for
Find the initial global optimal plan with the smallest fitness value from N plans
While i < Iterations **do**
  For j = 1 to N **do**
    Update the task offloading plans according to its search velocity
    Modify the offloading plans using the offloading decision of (12) and (13)
    Calculate the fitness value of each plan using (18)
  End for
  Find the global optimal plan with the smallest fitness value from N plans
  For j = 1 to N **do**
    Calculate the success value for each plan using (14)
  End for
  Obtain the success rate at this iteration using (15)
  Update the inertia weight using (16)
  Update each plan's search velocity using (17)
End while
Return the offloading plan X_{best}

The fitness function can measure the energy consumption of different offloading plans before the workflow deadline [44]. The higher the fitness value is, the greater the energy consumption. The calculation of the fitness contains two parts: the energy consumption of the offloading plan when the execution time is less than or equal to the deadline ($f_1 = 1$ and $f_2 = 0$) and when the execution time is greater than the deadline ($f_1 = 0$ and $f_2 = 1$). The value of deadline is between $deadline_{min}$ and $deadline_{max}$, which can be calculated using (19) and (20), respectively.

$$deadline_{min} = min\{T(X)\} \tag{19}$$

$$deadline_{max} = max\{T(X)\} \tag{20}$$

TABLE 5. Parameter settings.

Parameter	Value
Data sending power of the mobile device	P_s=0.1 W
Data receiving power of the mobile device	P_r=0.05 W
Computation power of the mobile device	P_m=0.5 W
Idle power of the mobile device	P_0=0.001 W
Average data transmission rate	R=100 Kb/s
CPU frequency of the mobile device	f_m=500 MHz
CPU frequency of the small virtual machine	f_{c1}=1000 MHz
CPU frequency of the middle virtual machine	f_{c2}=1300 MHz
CPU frequency of the large virtual machine	f_{c3}=1600 MHz

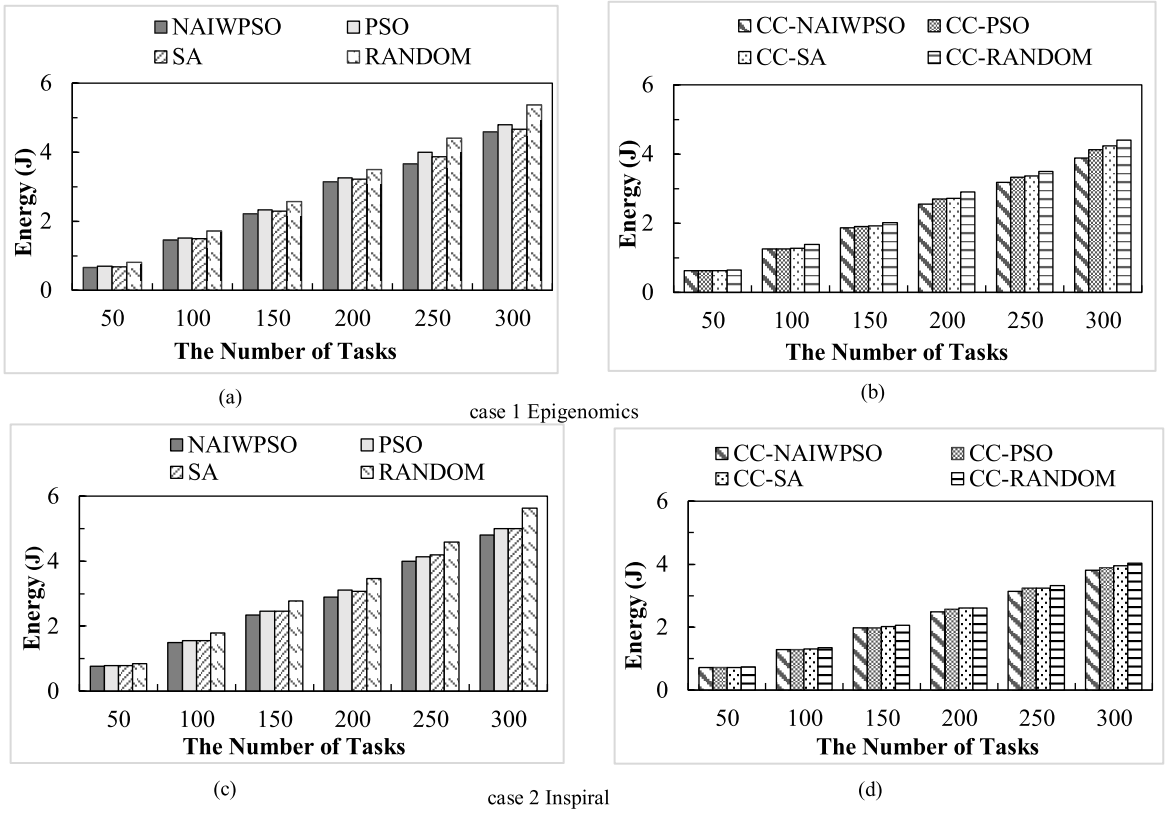


FIGURE 5. Energy consumption of mobile devices with different strategies under two workflow structures.

C. CC-NAIWPSO STRATEGY

According to the offloading decision model with the channel constraint, we present an energy-efficient and deadline-aware task offloading strategy named CC-NAIWPSO based on the NAIWPSO algorithm. The NAIWPSO algorithm allows the weights to adjust the particle velocities more precisely by improving the formula of the success value. Based on that, we apply the NAIWPSO algorithm to the offloading decision model to construct the CC-NAIWPSO, which can further reduce the energy consumption within the deadline and improve the convergence rate. The detailed flow of the CC-NAIWPSO strategy is presented as follows.

In CC-NAIWPSO, we first initialize the task offloading plans, the search velocity and other parameters (lines 1-8). While in the iterations, we update the offloading plans according to the search velocity (line 11). Then, the offloading plans are modified by the task offloading decision based on the channel constraint (line 12) and the fitness value

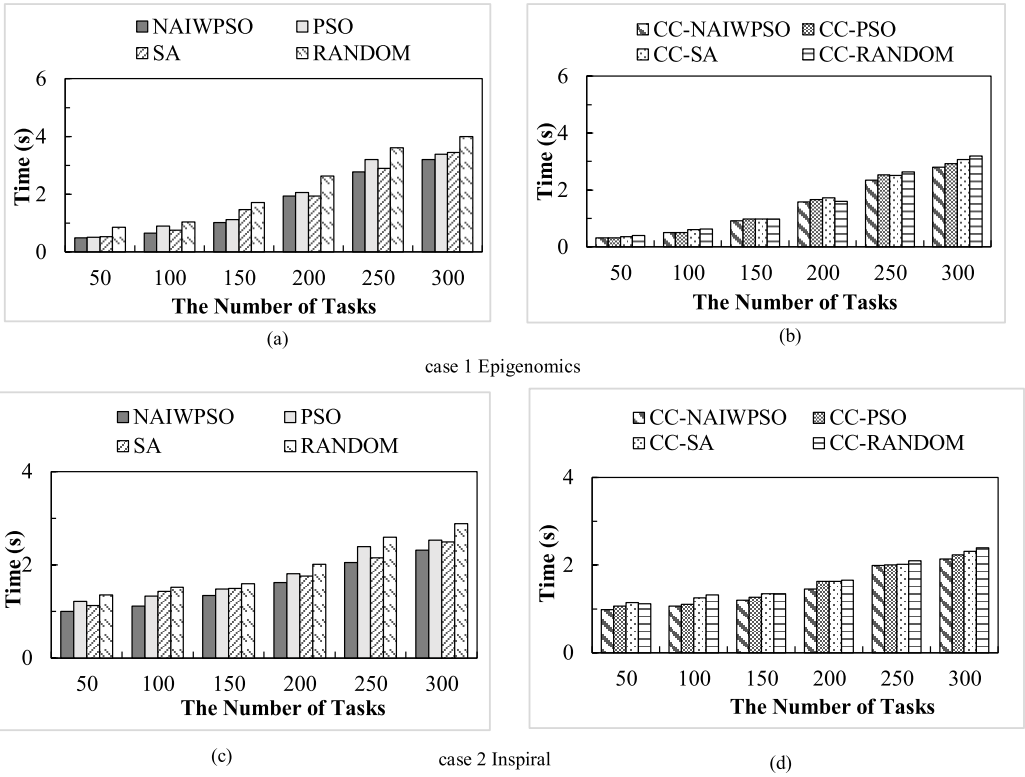


FIGURE 6. Workflow execution time with different strategies under two workflow structures.

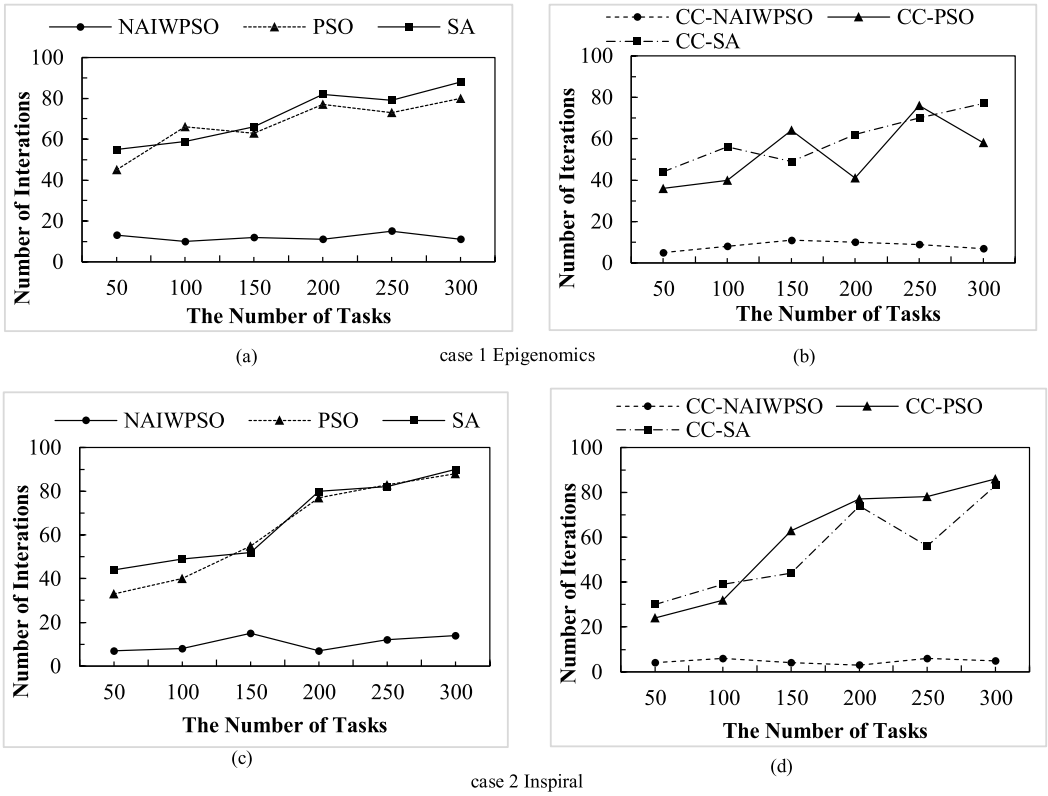


FIGURE 7. Number of iterations for different algorithms under two workflow structures.

of each offloading plan is calculated (line 13). Afterwards, we update the inertia weight and search velocity according to the NAIWPSO algorithm (lines 16-21). After a number of iterations, the final offloading plan Xbest is obtained (line 23). The NAIWPSO algorithm can avoid the problem of premature and local optimization, and it can achieve

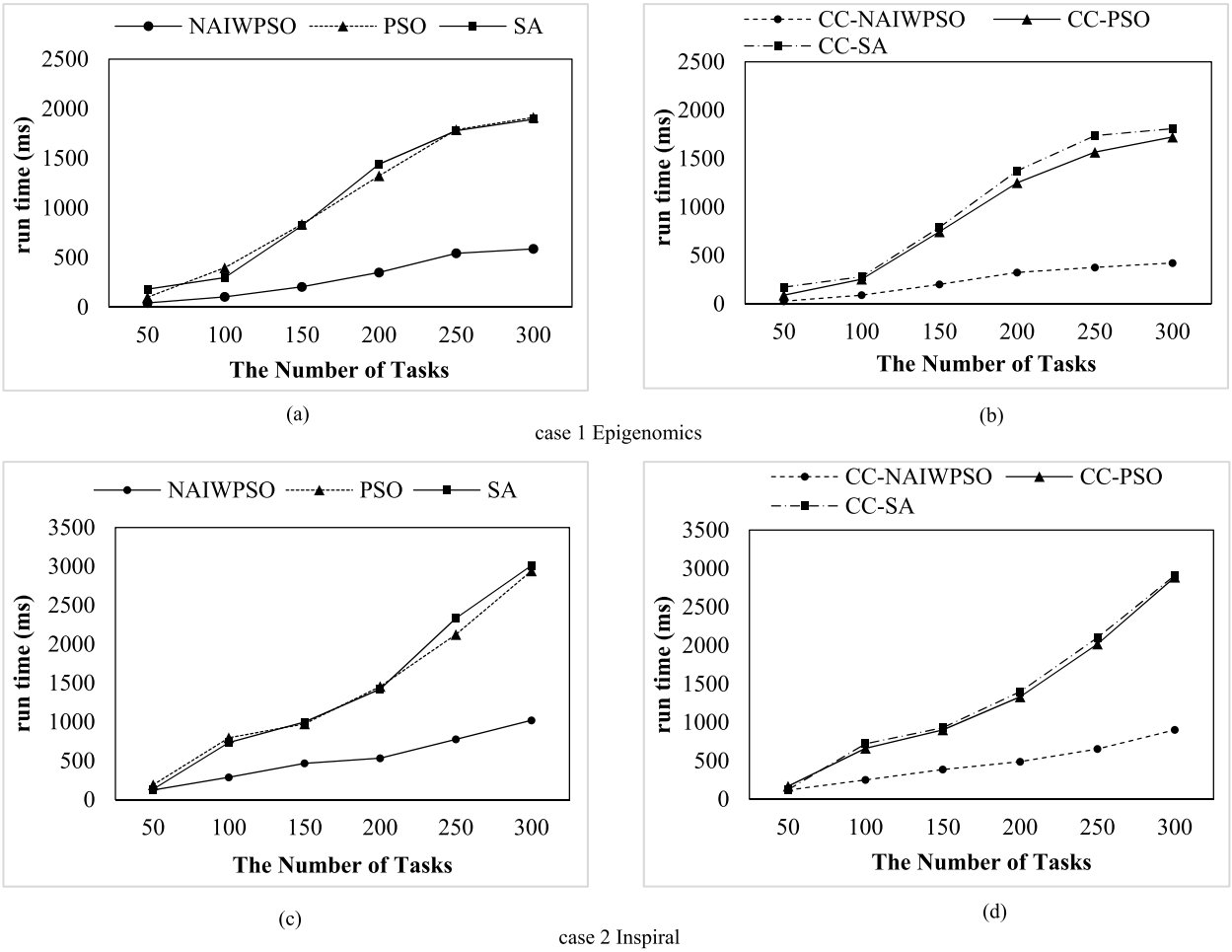


FIGURE 8. Running time of different algorithms under two workflow structures.

a stable convergence speed. Based on these results, our CC-NAIWPSO strategy can improve the convergence rate and generate task offloading plans that can greatly reduce the execution time of mobile workflows and the energy consumption of mobile devices.

VI. EXPERIMENTAL RESULTS

In this section, we first introduce the experimental environment and the setting of the experimental parameters. Then, we use two representative scientific workflows [45] to evaluate the effectiveness of our proposed task offloading strategy. Furthermore, the experimental results are analyzed in terms of the energy consumption of the mobile device and the execution time of mobile workflows. We also evaluate the convergence rate and running time of algorithms and the energy consumption of the mobile device under different deadlines. Finally, the results for the number of tasks which have been offloaded to cloud servers are also demonstrated.

A. EXPERIMENTAL ENVIRONMENT AND PARAMETER SETTINGS

The simulation environment runs on a PC with the following configurations: a dual-core CPU, 8GB of RAM, and the Microsoft Windows 10 OS. We use Java to deploy the CC-NAIWPSO strategy and other strategies. For each workflow structure, the number of tasks varies from 50 to 300. In different situations, the deadline is respectively set as the average time of executing the workflow 100 times. We create 3 virtual machines, namely, one small, one medium, and one large virtual machine. The workload of each task is in the range of 0-50 M cycles. The input and output data are in the range of 0-30 Kb [46]. $P_{BG} = 0.005$, and $P_{GB} = 0.005$ [43]. The parameters that are involved in the experiment are shown in Table 5 [47]. All experiments are repeated 100 times and the average results are obtained.

B. EXPERIMENTAL RESULTS ANALYSIS

The experimental results have shown that CC-NAIWPSO has great advantages in reducing the energy consumption and execution time. Specifically, in order to provide a fair comparison, we first compared NAIWPSO with three other representative offloading algorithms including the PSO algorithm [48], the simulated annealing (SA) algorithm [49] and the random (RANDOM) algorithm. In addition, we applied the offloading decision model with the channel constraint

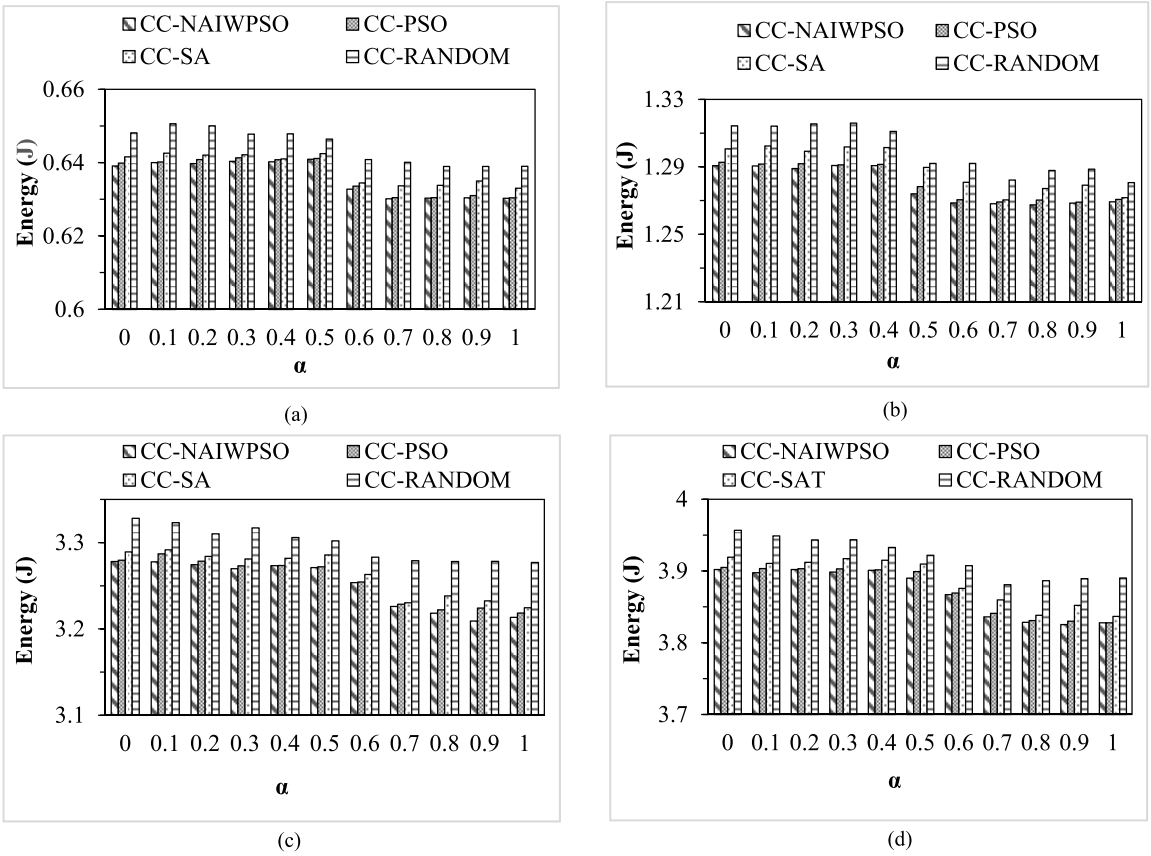


FIGURE 9. Energy consumption under different deadlines for epigenomics.

to these algorithms so that we can further compare our CC-NAIWPSO with CC-PSO, CC-SA, and CC-RANDOM.

1) ENERGY CONSUMPTION OF MOBILE DEVICES

The energy consumption of each strategy with the number of tasks varying from 50 to 300 under two workflow structures is shown in Fig. 5.

Figs. 5 (a) and (c) show that NAIWPSO achieves the best performance with the lowest energy consumption for two different workflows. It is able to find near-optimal offloading plans mainly because of the improvement of the adaptability of the inertia weight. As shown in Figs. 5 (b) and (d), with the channel constraint based offloading decision model, the energy consumption is further reduced by all strategies for different workflows. CC-NAIWPSO performs better than all others with the channel constraint. Therefore, we can conclude that CC-NAIWPSO can achieve the best performance in minimizing the energy consumption under different workflow structures.

2) EXECUTION TIME OF WORKFLOW APPLICATIONS

The execution time of two workflows using different strategies with the number of tasks varying from 50 to 300 is shown in Fig. 6. As mentioned above, the deadline is respectively set as the average time of executing the workflow 100 times under different situations.

From Figs. 6 (a) and (c), we can see that for the two different workflows, the workflow execution time with NAIWPSO is the lowest compared with the other algorithms. Clearly, the NAIWPSO algorithm can improve the efficiency of the workflow's execution. Furthermore, in Figs. 6 (b) and (d), CC-NAIWPSO performs the best compared with other strategies with the channel constraint. Therefore, we can conclude that CC-NAIWPSO improves the execution efficiency of the workflow applications while reducing the energy consumption.

3) CONVERGENCE RATE OF ALGORITHMS

The convergence rate is a critical measurement for the efficiency of the algorithm itself. In Fig. 7, the convergence rate of each strategy with the number of tasks varying from 50 to 300 under two workflow structures is presented. Note that since the convergence rate does not apply to the RANDOM algorithm, it is omitted here.

Figs. 7 (a) and (c) show that compared with two other traditional strategies (PSO and SA), the number of iterations for NAIWPSO is much lower and more stable under different workflows. Therefore, the NAIWPSO algorithm is more efficient when searching for the best solution. Furthermore, given

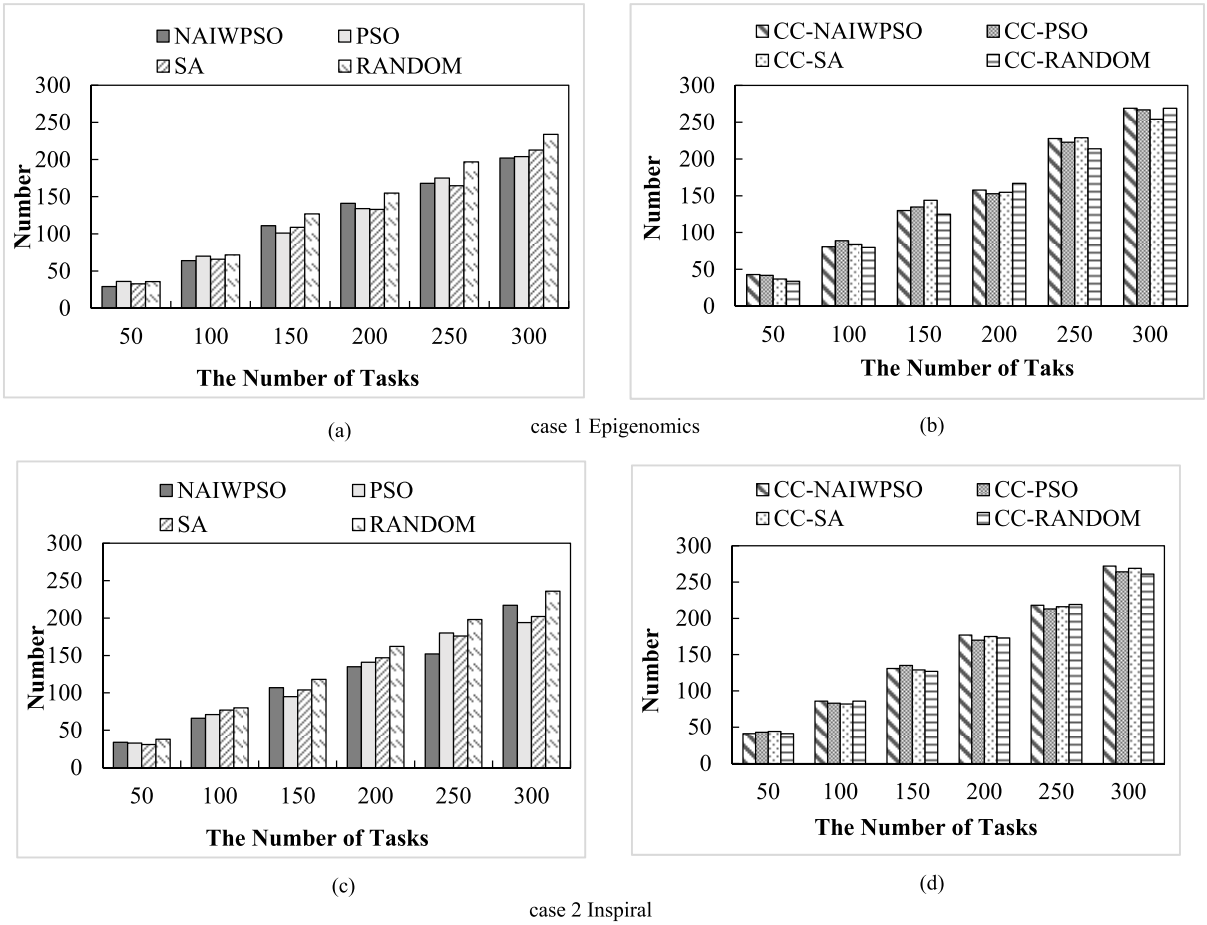


FIGURE 10. Number of tasks offloaded to cloud servers with different strategies under two workflow structures.

the offloading decision model with the channel constraint, CC-NAIWPSO keeps the convergence rate stable and further decreases the number of iterations compared with NAIWPSO, as shown in Figs. 7 (b) and (d). Similarly, the convergence rate has also been greatly reduced for CC-PSO and CC-SA compared with PSO and SA. Hence, our offloading decision model with the channel constraint can significantly improve the convergence rate. Therefore, we can conclude that CC-NAIWPSO has the highest efficiency and stability under all different cases.

4) RUNNING TIME OF ALGORITHMS

The running time of the algorithm directly reflects the efficiency of the algorithm. We compare the running time of different algorithms for various task numbers in mobile workflows. The running time is calculated from the beginning of the execution of the algorithm until it finally converges. Note that since the RANDOM strategy does not have an iterative process, we do not consider its running time.

In Fig. 8, the NAIWPSO and CC-NAWIPSO's running time are obviously lower than other strategies. This is because the two strategies are able to converge faster after the adoption of the NAIW. In addition, the running time of strategies with channel constraint is further reduced, which proves that the offloading decision model with the channel constraint is of significance to improving the run time of algorithms. Therefore, we can conclude that CC-NAIWPSO's running time keeps the lowest and it has the highest efficiency under all different cases.

5) ENERGY CONSUMPTION UNDER DIFFERENT DEADLINES

We use an epigenomics workflow as an example with task sizes of 50, 100, 250 and 300 to demonstrate the impacts of different deadlines on the energy consumption. The value of $deadline_{min} + \alpha(deadline_{max} - deadline_{min})$ is set as the abscissa, where α ranges from 0.1 to 1.0. Since in part 1) we have already shown that strategies with the channel constraint are better than the original ones, here we only include the results for strategies with the channel constraint.

As seen from Fig. 9, when the deadline is close to the minimum, all the strategies cannot have an appropriate offloading plan because the deadline is too restrictive. When the deadline moves from the minimum to the maximum, the energy consumption gradually decreases. The energy consumption of all strategies tends to be stable until the deadline reaches a certain range where all tasks are assigned to the virtual machines with

lower energy consumption, regardless of the four different strategies. Based on the results, we can conclude that our CC-NAIWPSO strategy can always achieve the lowest energy consumption under different deadlines.

6) NUMBER OF TASKS OFFLOADED TO CLOUD SEVERS

To demonstrate more details of our experiments, in Fig. 10, the number of tasks offloaded from the mobile to cloud servers are presented. The results have shown that generally more tasks are offloaded to the cloud servers by the strategies with channel constraint compared with those without. Meanwhile, we can also observe that for the ''offloading ratio'', namely the number of offloaded tasks divided by the number of total tasks, it is generally stable across all strategies with or without channel constraint.

VII. CONCLUSION AND FUTURE WORK

The energy consumption of mobile devices is a critical issue in mobile cloud computing. Meanwhile, due to the dynamic network state, it is a big challenge to ensure the satisfaction of deadline requirements. In this paper, to reduce the energy consumption of mobile devices while satisfying the deadline requirements of mobile workflow applications, we proposed an energy-efficient task offloading strategy CC-NAIWPSO based on the channel constraint. We formulated a task offloading decision model with the channel constraint and applied it with the NAIWPSO algorithm to create the CC-NAIWPSO strategy, which aims to achieve the near-optimal offloading plan. The experimental results have shown that our strategy can effectively reduce the energy consumption of the mobile devices while satisfying the deadlines of workflow applications.

In the future, we will try to measure the real-time data transmission rate to improve the generated task offloading plan. Meanwhile, we can also explore other optimization algorithms such as combining the PSO with other heuristic algorithms to further improve the efficiency of the strategy.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their critical and constructive comments and suggestions. Besides, F. A. and T. C. Authors thank Jia Xu and Ruimiao Ding deeply for their instructive suggestions and valuable comments.

REFERENCES

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, ''A survey of mobile cloud computing: Architecture, applications, and approaches,'' *Wireless Commun. Mobile Comput.*, vol. 13, no. 18, pp. 1587–1611, Dec. 2013.

[2] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, ''A survey of mobile cloud computing application models,'' *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, 1st Quart., 2014.

[3] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, ''Heterogeneity in mobile cloud computing: Taxonomy and open challenges,'' *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 369–392, 1st Quart., 2014.

[4] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, ''To offload or not to offload? The bandwidth and energy costs of mobile cloud computing,'' in *Proc. 32nd IEEE INFOCOM Conf.*, Turin, Italy, Apr. 2013, pp. 1285–1293.

[5] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad, ''Energy efficient computational offloading framework for mobile cloud computing,'' *J. Grid Comput.*, vol. 13, no. 1, pp. 1–18, Mar. 2015.

[6] P. V. Krishna, S. Misra, V. Saritha, D. N. Raju, and M. S. Obaidat, ''An efficient learning automata based task offloading in mobile cloud computing environments,'' in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, pp. 1–6.

[7] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, ''Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing,'' *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[8] W. Li, J. Cao, K. Hu, J. Xu, and R. Buyya, ''A trust-based agent learning model for service composition in mobile cloud computing environments,'' *IEEE Access*, vol. 7, pp. 34207–34226, 2019.

[9] Z. Kuang, S. Guo, J. Liu, and Y. Yang, ''A quick-response framework for multi-user computation offloading in mobile cloud computing,'' *Future Gener. Comput. Syst.*, vol. 81, pp. 166–176, Apr. 2018.

[10] X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao, ''Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing,'' *IEEE Access*, vol. 6, pp. 45725–45739, 2018.

[11] X.-F. Liu, Z.-H. Zhan, J. D. Deng, Y. Li, T. L. Gu, and J. Zhang, ''An energy efficient ant colony system for virtual machine placement in cloud computing,'' *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 113–128, Feb. 2018.

[12] B. Hayes, ''Cloud computing,'' *Commun. ACM*, vol. 51, no. 7, pp. 9–11, Jul. 2008.

[13] P. Singh, M. Dutta, and N. Aggarwal, ''A review of task scheduling based on meta-heuristics approach in cloud computing,'' *Knowl. Inf. Syst.*, vol. 52, no. 1, pp. 1–51, Apr. 2017.

[14] L. A. Tawalbeh, R. Mehmood, E. Benkhlifa, and H. Song, ''Mobile cloud computing model and big data analysis for healthcare applications,'' *IEEE Access*, vol. 4, pp. 6171–6180, 2016.

[15] D. Huang, T. Xing, and H. Wu, ''Mobile cloud computing service models: A user-centric approach,'' *IEEE Netw.*, vol. 27, no. 5, pp. 6–11, Sep./Oct. 2013.

[16] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, ''A survey of mobile cloud computing application models,'' *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, Jul. 2013.

[17] M. Mukherjee, R. Matam, L. Shu, L. A. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, ''Security and privacy in fog computing: Challenges,'' *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[18] M. Peng, S. Yan, K. Zhang, and C. Wang, ''Fog-computing-based radio access networks: Issues and challenges,'' *IEEE Netw.*, vol. 30, no. 4, pp. 46–53, Jul./Aug. 2016.

[19] Q. Huang, Y. Yang, and L. Wang, ''Secure data access control with ciphertext update and computation outsourcing in fog computing for Internet of Things,'' *IEEE Access*, vol. 5, pp. 12941–12950, 2017.

[20] Y. Hao, C. Min, H. Long, M. S. Hossain, and A. Ghoniem, ''Energy efficient task caching and offloading for mobile edge computing,'' *IEEE Access*, vol. 6, pp. 11365–11373, 2018.

[21] F. Wang, J. Xu, X. Wang, and S. Cui, ''Joint offloading and computing optimization in wireless powered mobile-edge computing systems,'' *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2017.

[22] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, ''Performance guaranteed computation offloading for mobile-edge cloud computing,'' *IEEE Wireless Commun. Lett.*, vol. 6, no. 6, pp. 774–777, Dec. 2017.

[23] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, ''Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks,'' *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[24] Z. Wei and H. Jiang, ''Optimal offloading in fog computing systems with non-orthogonal multiple access,'' *IEEE Access*, vol. 6, pp. 49767–49778, 2018.

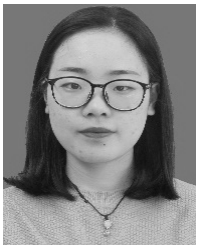
[25] W.-L. Zhang, B. Guo, Y. Shen, D.-G. Li, and J.-K. Li, ''An energy-efficient algorithm for multi-site application partitioning in MCC,'' *Sustain. Comput., Inform. Syst.*, vol. 18, pp. 45–53, Jun. 2018.

[26] N. Chen, X. Fang, and X. Wang, ''A cloud computing resource scheduling scheme based on estimation of distribution algorithm,'' in *Proc. 2nd Int. Conf. Syst. Inf.*, Shanghai, China, Nov. 2014, pp. 304–308.

[27] X. Lin, Y. Wang, Q. Xie, and M. Pedram, ''Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment,'' *IEEE Trans. Services Comput.*, vol. 8, no. 2, pp. 175–186, Dec. 2014.

[28] T. Li, B. Y. Wang, and S. O. Computer, ''Workflow energy-efficient scheduling algorithm in cloud environment with QoS constraint,'' *Comput. Sci.*, vol. 45, no. 6A, pp. 304–309, Jun. 2018.'

[29] M. A. Arfeen, K. Pawlikowski, and A. Willig, "A framework for resource allocation strategies in cloud computing environment," in *Proc. 35th Comput. Softw. Appl. Conf. Workshops*, Munich, Germany, Jul. 2011, pp. 261–266.
[30] W. J. Shi, J. G. Wu, and Y. C. Luo, "Fast and efficient scheduling algorithms for mobile cloud offloading," *Comput. Sci.*, vol. 45, no. 4, pp. 94–99, Apr. 2018.
[31] X. Wang and X. Liu, "Clouding computing resource scheduling based on double fitness dynamic genetic algorithm," *Comput. Eng. Des.*, vol. 39, no. 5, pp. 1372–1376, May 2018.
[32] C. Ju, X. Zhao, and M. Wang, "Application of improved genetic algorithm in optical scheduling of cloud computing resources," *Softw. Guide*, vol. 17, no. 4, pp. 45–46, Apr. 2018.
[33] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015.
[34] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocess. Microsyst.*, vol. 26, no. 8, pp. 363–371, 2002.
[35] X. Chen, J. W. Xu, and D. Long, "Resource scheduling algorithm of cloud computing based on ant colony optimization-shuffled frog leading algorithm," *J. Comput. Appl.*, vol. 38, no. 6, pp. 1670–1674, Jun. 2018.
[36] C. Y. Liu and W. W. Yang, "A multi-objective task scheduling based on genetic and particle swarm optimization algorithm for cloud computing," *Comput. Technol. Develop.*, vol. 27, no. 2, pp. 56–59, Feb. 2017.
[37] B. Wang and X. Zhang, "Task scheduling algorithm based on particle swarm optimization genetic algorithms in cloud computing environment," *Comput. Eng. Appl.*, vol. 51, no. 6, pp. 84–88, Jun. 2015.
[38] J. R. Xu and H. J. Zhu, "Coevolutionary genetic algorithm of cloud workflow scheduling based on adaptive penalty function," *Comput. Sci.*, vol. 45, no. 8, pp. 105–112, Aug. 2018.
[39] G.-Y. Cai and E.-Q. Dong, "Comparison and analysis of generation algorithm and ant colony optimization on TSP," *Comput. Eng. Appl.*, vol. 43, no. 10, pp. 96–98, Oct. 2007.
[40] C. J. He and Z. J. Bai, "Task scheduling based on improved ant colony algorithm in cloud environment," *Comput. Technol. Develop.*, vol. 28, no. 12, pp. 13–16, Dec. 2018.
[41] X. J. Li, J. Xu, F. Wang, E. Z. Zhu, and L. Wu, "Energy aware task scheduling algorithm in cloud workflow system," *Pattern Recognit. Artif. Intell.*, vol. 40, no. 2, pp. 364–377, Sep. 2016.
[42] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul./Sep. 2018.
[43] X. Liu, J. Li, and Z. Yang, "A task collaborative execution policy in mobile cloud computing," *Chin. J. Comput.*, vol. 40, no. 2, pp. 364–377, Feb. 2017.
[44] N. Netjinda, B. Sirinaovakul, and T. Achalakul, "Cost optimal scheduling in IaaS for dependent workload with particle swarm optimization," *J. Supercomput.*, vol. 68, no. 3, pp. 1579–1603, Jun. 2014.
[45] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proc. 3rd Workshop Workflows Support Large-Scale Sci.*, Austin, TX, USA, Nov. 2008, pp. 1–10.
[46] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *Proc. 32nd IEEE INFOCOM Conf.*, Turin, Italy, Apr. 2013, pp. 190–194.
[47] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.
[48] R. C. Eberhart and Y. H. Shi, "Particle swarm optimization: Developments, applications and resources," in *Proc. Congr. Evol. Comput.*, Seoul, South Korea, May 2001, pp. 81–86.
[49] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.



YINGJIE WANG is currently pursuing the bachelor's degree with Anhui University, China. Her research interests include mobile cloud computing and workflow systems.

LEI WU received the Ph.D. degree from Anhui University, China, in 2018, where she is currently a Lecturer with the School of Computer Science and Technology. Her major research interests include workflow systems and cloud computing, scheduling, and optimization.

XIUSHENG YUAN is currently pursuing the bachelor's degree with Anhui University, China. His research interests include mobile cloud computing and workflow systems.

XIAO LIU received the master's degree in management science and engineering from the Hefei University of Technology, Hefei, China, in 2007, and the Ph.D. degree in computer science and software engineering from the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, VIC, Australia, in 2011. He is currently a Senior Lecturer with the School of Information Technology, Deakin University, Melbourne. He has published over 90 papers in the area of software engineering and service computing including journals, such as the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and the IEEE TRANSACTIONS ON SERVICE COMPUTING.

XUEJUN LI (M'18) received the Ph.D. degree from Anhui University, China, in 2008, where he is currently a Professor with the School of Computer Science and Technology. His major research interests include workflow systems, cloud computing, and intelligent software.