micromachines

MDPI

Article

An Efficient Computation Offloading Strategy with Mobile Edge Computing for IoT

Juan Fang *, Jiamei Shi, Shuaibing Lu *, Mengyuan Zhang and Zhiyuan Ye

Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China; shijiamei@emails.bjut.edu.cn (J.S.); mengyuanzhang@emails.bjut.edu.cn (M.Z.); yezy@emails.bjut.edu.cn (Z.Y.)
* Correspondence: fangjuan@bjut.edu.cn (J.F.); lushuaibing@bjut.edu.cn (S.L.); Tel.: +86-139-1129-6256 (J.F.)

Abstract: With the rapidly development of mobile cloud computing (MCC), the Internet of Things (IoT), and artificial intelligence (AI), user equipment (UEs) are facing explosive growth. In order to effectively solve the problem that UEs may face with insufficient capacity when dealing with computationally intensive and delay sensitive applications, we take Mobile Edge Computing (MEC) of the IoT as the starting point and study the computation offloading strategy of UEs. First, we model the application generated by UEs as a directed acyclic graph (DAG) to achieve fine-grained task offloading scheduling, which makes the parallel processing of tasks possible and speeds up the execution efficiency. Then, we propose a multi-population cooperative elite algorithm (MCE-GA) based on the standard genetic algorithm, which can solve the offloading problem for tasks with dependency in MEC to minimize the execution delay and energy consumption of applications. Experimental results show that MCE-GA has better performance compared to the baseline algorithms. To be specific, the overhead reduction by MCE-GA can be up to 72.4%, 38.6%, and 19.3%, respectively, which proves the effectiveness and reliability of MCE-GA.

Keywords: mobile edge computing; computation offloading; offloading strategy; genetic algorithm

Citation: Fang, J.; Shi, J.; Lu, S.; Zhang, M.; Ye, Z. An Efficient Computation Offloading Strategy with Mobile Edge Computing for IoT. *Micromachines* **2021**, *12*, 204. https://doi.org/10.3390/mi12020204

Academic Editors: Roberto Cavicchioli and Paolo Burgio

Received: 6 January 2021
Accepted: 15 February 2021
Published: 17 February 2021

1. Introduction

Mobile cloud computing (MCC), the Internet of Things (IoT), and artificial intelligence (AI) are evolving rapidly, so new applications such as video analysis, virtual reality, and intelligent vehicles are constantly emerging. Moreover, applications may require real time processing such as drone flight control applications, Augmented Reality (AR)/Virtual Reality (VR), and online gaming, with requirements of latencies below a few tens of milliseconds [1]. Cisco notes that by 2021, IoT devices will dominate connectivity. Of the 27.1 billion connected devices in the world, the Internet of Things will reach 13.7 billion [2]. Although MCC can provide centralized computing resources, due to a large amount of data to be transmitted, the transfer of all IoT sensed mobile terminal data to the cloud data center will bring huge crowding pressure and high delay to the network, thus seriously affecting the quality of user experience.

To solve the above problems, mobile edge computing (MEC) emerged. MEC refers to the formation of the edge cloud on the edge servers close to mobile devices in the IoT. It provides users with computing and storage resources which reduces the share of network resources, the energy consumption of UEs, and the network delay [3–5]. With the help of MEC, the intelligence of IoT can be improved and the IoT can be rooted in every vertical industry. At present, MEC has attracted extensive attention in the industry. MEC has many application scenarios, among which the typical application scenarios are as follows. (1) There are some local edge application services related to network information function opening and network access, such as positioning and navigation system based on the wireless network [6]. (2) There are some high-definition video acceleration servers. The powerful computing capacity of MEC can greatly reduce the response delay of users while watching videos to ensure the smoothness [7,8]. (3) There are some network edge

application services for industrial IoT, Internet of vehicles, and other application scenarios, which have ultrahigh demand for delay, reliability, and computing performance [9,10]. (4) There are many computationally intensive tasks in the blockchain that need to be offloaded to the edges [11,12]. By combining the features of decentralization, consensus mechanism and peer-to-peer interconnection of the block chain technology, the edge nodes can be trusted and authenticated to improve the security of the service and system [13].

MEC is a promising enabler of future Internet such as the fifth generation (5G) [14], enabling multiple future 5G applications and network services such as IoT applications, haptic Internet, AR/VR use cases, or remote driving [15]. Computation-intensive applications, such as AR and VR, require a lot of computation in a very short period of time, which need very high computing power. For example, AR is a technology that uses additional information generated by computer to enhance or extend the user's view of the real world. Additionally, VR is a computer simulation technology that uses a computer to fuse multiple information and entity behavior to simulate three-dimensional dynamic views. Both of them need to collect real-time information about the status of users including locations and orientations, and then perform calculations. The MEC server can provide rich computing resources and storage resources, buffer audio and video content according to the location information, determine the push content, send it to the user or quickly simulate the 3D dynamic view, and interact with the user, thus greatly improving the user experience.

In MEC, the improvement of mobile application performance is largely dependent on efficient task offloading decisions. Therefore, offloading decision-making has been widely concerned by scholars in recent years. Yang et al. [16] considered the capacity constraints of lead-time and backhaul links and the maximum delay constraints of users and proposed an effective unloading scheme to minimize the total network energy consumption. Zhang et al. [17] proposed a computation offloading scheme for energy perception by weighing energy consumption and time delay, and introduced the residual energy of the smart device battery into the definition of the weighted factor of energy consumption and delay, which effectively reduces the total system consumption. However, the above literature failed to allocate the limited wireless and computing resources reasonably. Tong et al. [18] proposed an adaptive task offloading and resource allocation algorithm in MEC environment. The algorithm used the Deep Reinforcement Learning (DRL) method to determine whether the task needs to be offloaded and allocate computing resources for the task. However, the deep reinforcement learning method also has some defects. It is difficult to adjust parameters, and the training time is long. There are also many solutions for offloading problems under different environmental scenarios from the perspective of optimizing energy consumption. Zhang et al. [19] adopted artificial fish swarm algorithm to design the offloading strategy of energy consumption optimization under the constraint of time delay. This strategy considered the link condition in the task data transmission network and effectively reduced the equipment energy consumption, but it had the defect of high algorithm complexity. In the scenario of multiple resources, Xu et al. [20] designed a task scheduling algorithm of energy consumption minimization particle swarm optimization for multiple resources matching to reduce energy consumption of edge terminal equipment. Zhao et al. [21] proposed a privacy perception computing offloading algorithm based on Lyapunov optimization theory. Liu et al. [22] studied deep learning task offloading. In order to better deploy deep learning applications and optimize network power consumption, a set of sparse beamforming framework based on mixed L1/L2 norms was proposed. However, the above literature does not take into account the delay. At present, some researches regard the computing task of mobile terminal offloading as composed of several subtasks with dependencies among each other and consider the fine-grained task offloading. Ding et al. [23] studied the code-oriented partition computing offload strategy and proposed an offloading strategy to determine the user's execution location and minimize the system overhead. However, the parallelism of the tasks is not considered.

Different from the above methods, our paper designs an efficient computation offloading strategy in the auxiliary network with multiple MEC servers, which considers

the partition of users' applications. It realizes fine-grained task offloading scheduling. Our study aims at minimizing the overhead of generated applications by improving the parallel computing capacity of MEC servers. The main contributions of this paper are shown as follows.

1. We consider the heterogenous properties of MEC and the resource limitation of UEs and MEC servers. We jointly optimize the execution delay and energy consumption of applications generated by UEs.
2. We consider fine-grained task computation offloading of fine-grained tasks, which have dependencies, model the user-generated mobile application as a directed acyclic graph, and make the parallel processing of tasks possible.
3. In order to reduce the overhead of applications generated by UEs and improve the utilization rate of system resources, we proposed a multi-population coevolutionary elite genetic algorithm (MCE-GA) to solve resource allocation and task scheduling problem. By simulation experiments, we verify the effectiveness of the MCE-GA algorithm.

The rest of this paper is organized as follows. Section 2 describes the system model and formulates the problem of computing offloading. Section 3 describes the proposed computation offloading algorithm. Section 4 describes the proposed simulation analysis method and compares it with other algorithms. Section 5 summarizes our study.

2. System and Computation Model

In this section, we describe the system and computation model of MEC in detail.

2.1. System Model

We consider the scenario of multiple users and multiple MEC servers. In the whole network, UEs are connected through base stations and wireless channels. Multiple MEC servers are deployed beside the base stations to provide computing services for UEs. Cloud servers are located on top of the core network far from the UEs. Compared with the unpredictable delay and long transmission distance caused by mobile cloud computing technology used by UEs to offload computing to cloud servers, MEC can provide computing services for UEs more quickly and efficiently and relieve the pressure on core networks [24]. In our work, we only consider the computation offloading problem between the user layer and the edge layer, where the user chooses to offload the task to the local or MEC server.

The overall network of system is shown in Figure 1. In this system, the top end is the cloud server communicating with the base station through a switch. There are multiple base stations in the whole network. A number of small MEC servers are deployed near each base station, and the communication can be conducted between MEC servers. On the terminal smart device side, the tasks can be executed directly on the local server or sent via the data transfer unit to a MEC server in its area for remote computation.

The whole network of system is composed of several small areas, which are independent of each other. We consider the computation offloading situation in a small area and have the following assumptions. There are U UEs and M MEC servers in an area. UE_i ($i \in \{1, 2, \ldots, U\}$) offloads computing tasks to MEC_j ($j \in \{1, 2, \ldots j, \ldots M\}$) through wireless links communication. The wireless links are orthogonal, so links do not interfere with each other. What's more, UE_i communicates with all MEC servers that can be connected and perceives the computing resources of these MEC servers. UE_i can choose to compute locally or offload MEC servers in this area. Let f_i and F_j denote the computing capacity of UE_i and MEC_j, respectively. The main notations involved in this paper are given in Table 1.

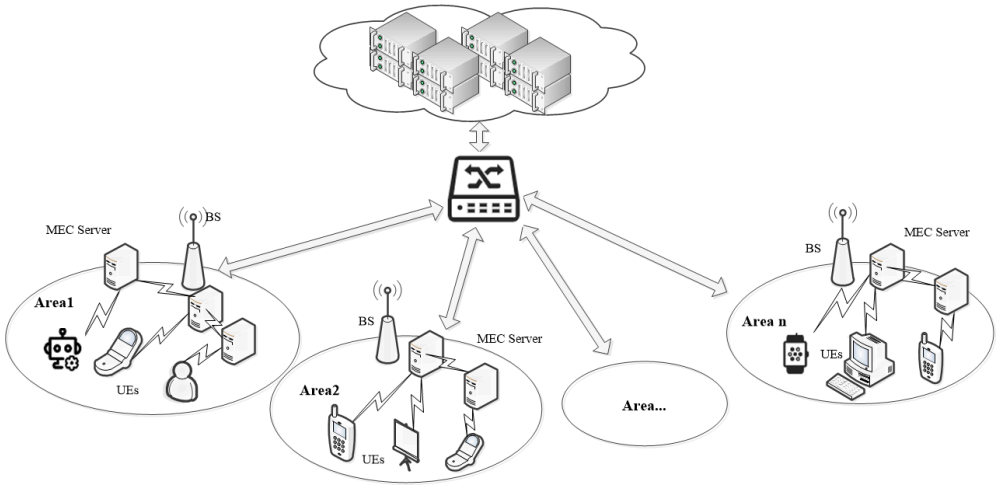


Figure 1. Overall network of system.

Table 1. Notations.

Symbol	Description
UE	User equipment
MEC	MEC servers
f_i	Computing capacity of UE_i
F_j	Computing capacity of MEC_j
A_i	Application generated by UE_i
v_k	Task of Application
w_k	Workload of v_k
d_k	Data size of v_k
σ_k	The ratio of the output/input data size
τ_k	Maximum delay a task can tolerate
B	Transmission bandwidth
N	Background noise
p	The transmission power of UE_i
$d_{i,j}$	The distance between UE_i and MEC_j
$d_{j',j}$	The distance of MEC servers
h	The channel gain
k_i	The coefficient factor of $UE_i's$ chip architecture
θ	The channel fading coefficient

2.2. Application Model

We assume that the application generated by UE is composed of several tasks with dependency. We first model it using a directed acyclic graph (DAG) to achieve fine-grained task offloading scheduling. Based on that, we analyze the probability of the parallel processing for tasks, which can speed up the execution efficiency and corresponds to a more realistic scenario. Figure 2 shows an example of the application model produced by UE.

We model the application as a DAG graph, where UE_i generates application A_i. So UE_i can also be expressed as $A_i = \{V_i, E_i\}$, where V_i is the set of generated tasks, and E_i is the dependency relationship between tasks. We define a task generated by UE as v_k where $v_k \in V_i$. For example, in Figure 2, there are v_0, $v_1 \in E_i$, and there is a directed edge from v_0 to v_1 in A_i, thus v_1 must be executed after v_0. Moreover, we define $v_k = \{w_k, d_k, \sigma_k, \tau_k\}$. w_k refers to the total workload of v_k, which indicates the number of CPU clocks cycles required to execute v_k. d_k indicates data size of the task and σ_k refers to the ratio of the output/input data sizes of tasks generated by UE_i, τ_k presents the maximum delay a task can tolerate. We use a binary variable $I_{k,i,j} \in \{0, 1\}$ to indicate whether the task was offloaded to the

MEC_j. We use $I_{k,i,0} = 1$ to indicate that the task was executed locally. Since we assume one task can only be executed on one position, we set $\sum_{j=0}^{M} I_{k,i,j} = 1$.

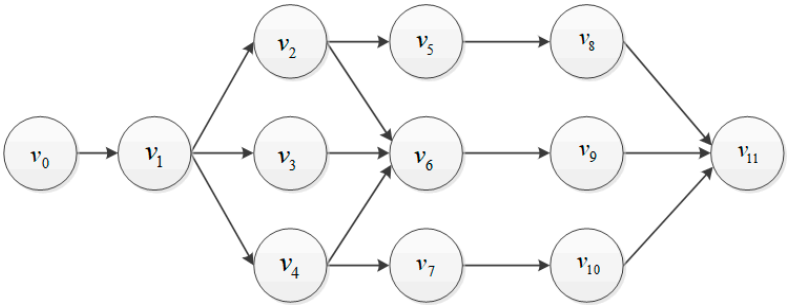


Figure 2. The application model generated by user equipment (UE).

2.3. Communication Model

In the whole system, there are mainly three kinds of communication depending on user's offloading location. (1) UE_i offloads v_k to MEC_j. (2) MEC_j receives the computation result of v_k from $MEC_{j'}$. (3) UE_i receives the result of v_k from MEC_j.

According to Rayleigh fading channel model [25], the rate of UE_i to transmit v_k to MEC_j can be defined

$$r_{i,j,v} = Blog\left(1 + \frac{p_{i,j}h_{i,j}}{d_{i,j}^{\theta}N}\right), \tag{1}$$

where B represents the transmission bandwidth, $p_{i,j}$ represents the transmission power from UE_i to MEC_j, $h_{i,j}$ represents the channel gain between UE_i and MEC_j, $d_{i,j}$ represents the distance between UE_i and MEC_j, θ represents the path loss exponent, and N represents Gaussian noise.

The rate of UE_i to receive the result of v_k from MEC_j can be defined as

$$r_{j,i,res} = Blog\left(1 + \frac{p_{j,i}h_{j,i}}{d_{j,i}^{\theta}N}\right). \tag{2}$$

Similarly, $p_{j,i}$ represents the receiving power of UE_i from MEC_j, $h_{j,i}$ represents channel gain, and $d_{j,i}$ represents the distance between MEC_j and UE_i.

Moreover, when MEC_j executes task v_k, if v_{k-1} is executed in $MEC_{j'}$, then the transmission rate of receiving the result of the precursor task from $MEC_{j'}$ is defined as

$$r_{j',j,pre} = Blog\left(1 + \frac{p_{j',j}h_{j',j}}{d_{j',j}^{\theta}N}\right), \tag{3}$$

where $p_{j',j}$ represents transmission power of $MEC_{j'}$, $h_{j',j}$ represents channel gain, and $d_{j',j}$ represents distance between $MEC_{j'}$ and MEC_j.

2.4. Computation Model

In this subsection, we describe the computation model in detail. Computing offloading is performed either locally or by MEC servers. We consider fine-grained task offloading, so the dependencies between tasks are also needed to be considered. The dependencies between tasks mean that all precursors of task must have finished executing before it can begin executing. Obviously, more complex dependencies between tasks will increase the complexity of task computation offloading decision. Next, we give the definition of ready time as follows.

Definition 1 (Ready Time). Ready Time denotes the total wait time required for task v_k to start execution.

(1) Local Computation Model

According to the task model defined above, w_k is the workload of the task v_k, which is the total CPU cycles needed to execute the task. f_i is the computing ability of UE_i. The local execution delay of task v_k for UE_i is defined as

$$T^l_{k,i,exe} = \frac{w_k}{f_i}. \tag{4}$$

In addition, we assume that UE_i have limited resources and can only execute one task at a time, so UE_i have a waiting queue to store tasks that need to be computed locally. A wait delay occurs when multiple tasks of UE_i are offload locally. The waiting delay of task v_k is the total delay of the completion delay of the tasks in the waiting queue that arrived earlier than task v_k.

We use $FT_{pre,i}$ to represent the completion delay of the precursor of task v_k, $FT^{res}_{pre,i}$ to represent the return delay after the precursor for task v_k was completed, and T^l_{wait} to represent the local execution delay of task v_k. $FT^{res}_{pre,i}$ can be defined as

$$FT^{res}_{pre,i} = \frac{d_{pre} * \sigma_{pre}}{r_{j,i,pre}}, \tag{5}$$

where d_{pre} represents data size of the precursor of task v_k, σ_{pre} represents ratio of the output data size to the input data size of task v_{pre}, and $r_{j,i,pre}$ represents the rate from MEC_j to UE_i.

Thus, the ready time of v_k can be defined as

$$RT^l_{k,i} = \max_{pre \in pred(v_k)} (FT_{pre,i} + FT^{res}_{pre,i}, \quad T^l_{wait}), \tag{6}$$

where T^l_{wait} indicates the total waiting time required for task v_k to start execution.

Therefore, the total delay for task v_k of UE_i to execute locally can be defined as

$$FT^l_{k,i} = T^l_{k,i,exe} + RT^l_{k,i}. \tag{7}$$

According to [26], the total energy consumption for task v_k of UE_i to execute locally can be defined as

$$E^l_{k,i} = k_i w_i f_i^2, \tag{8}$$

where k_i is the coefficient factor of UE_i's chip architecture.

(2) MEC Server Computation Model

When UE_i chooses to offload to MEC server for computation, the transmitting delay of task v_k from UE_i to MEC_j is defined as

$$T^j_{k,i,trs} = \frac{d_k}{r_{i,j,k}}. \tag{9}$$

The computational time of task v_k executed at MEC_j is defined as

$$T^j_{k,i,exe} = \frac{w_k}{F_j}, \tag{10}$$

where d_k is the data size of v_k, $r_{i,j,k}$ is the transmission rate from UE_i to MEC_j, and F_j is the computing ability of MEC_j.

Similarly, the ready time for the task v_k needs to be considered. When the task v_k chooses to offload on MEC_j, the ready time involves waiting for the precursor tasks of task v_k to finish executing and passing the result back to MEC_j, waiting for the transmitting delay of task v_k from UE_i to MEC_j. In addition, we suppose that the MECs also have limited computing resources and execute one task at a time, so the ready time also includes

the sum of the completion delays of tasks in the MEC_j waiting queue that arrive before the current task. Figure 3 shows an example of the task queue model.

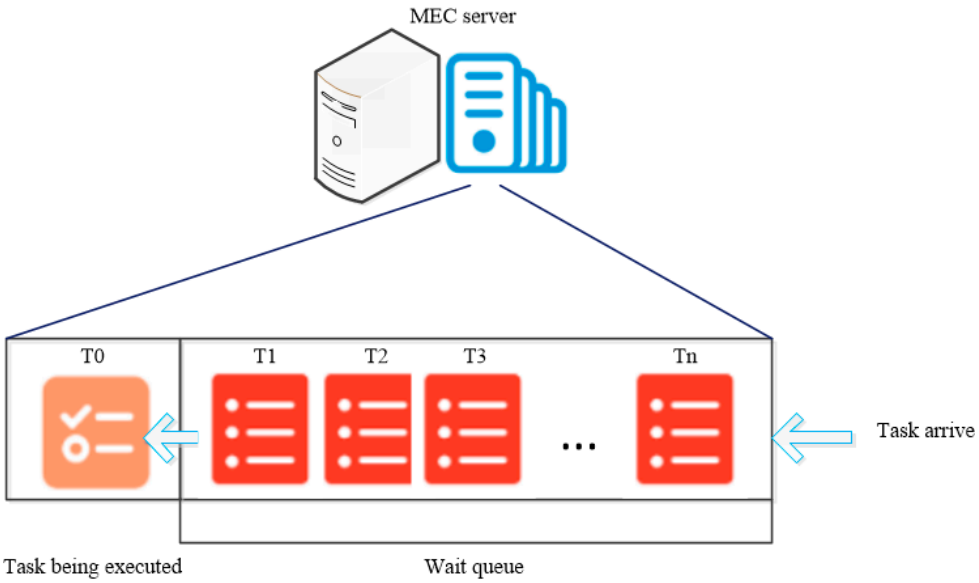


Figure 3. An example of the Task Queue model.

Thus, the ready time for task v_k executed on MEC_j is expressed as $RT^j_{k,i}$ and can be defined

$$RT^j_{k,i} = \max_{pre \in pred(v_k)} (FT_{pre,i} + FT^{res}_{pre,i},\ T^j_{wait},\ T^j_{k,i,trs}), \tag{11}$$

where $FT_{pre,i}$ is the delay of the precursor of v_k, $FT^{res}_{pre,i}$ is the result return time of the precursor task of v_k, and T^j_{wait} is the queue wait delay in MEC_j.

Thus, the total delay of task v_k executed at MEC_j can be defined as

$$FT^j_{k,i} = T^j_{k,i,exe} + RT^j_{k,i}. \tag{12}$$

The total energy consumption of task v_k executed at MEC_j can be defined as

$$E^j_{k,i} = p_{i,j} * T^j_{k,i,trs} + p_{i,j} * T^j_{k,i,res}. \tag{13}$$

2.5. Problem Formulation

In this subsection, we formulate the computation offloading problem to optimize the overhead of applications generated by UE_i.

The total delay of UE_i can be formulated as

$$T_{i,total} = \sum_{k=1}^{|v_i|} ((1 - \sum_{j=1}^{M} I_{k,i,j}) FT^l_{k,i} + \sum_{j=1}^{M} I_{k,i,j} FT^j_{k,i}). \tag{14}$$

The total energy of UE_i can be formulated as

$$E_{i,total} = \sum_{k=1}^{|v_i|} ((1 - \sum_{j=1}^{M} I_{k,i,j}) E^l_{k,i} + \sum_{j=1}^{M} I_{k,i,j} E^j_{k,i}). \tag{15}$$

Some tasks require low latency and some require low energy consumption. In order to consider these two consumptions, we draw on the work of [27] and define the total overhead of UE_i as

$$C_i = \beta * E_{i,total} + (1 - \beta) * T_{i,total}, \tag{16}$$

where $\beta \in [0,1]$ represents the tradeoff parameter of delay and energy.

In order to minimize the execution overhead of the application generated by UE_i, for UE_i, we can formulate the problem as

$$\begin{aligned} P1: \quad & \min_{I_i} C_i \\ s.t. \quad & C1: T_{i,total} \leq \sum_{k=1}^{|v_i|} \tau_k \\ C2: \quad & (v_k, v_{k+1}) \in E_i,\ v_k, v_{k+1} \in V_i \\ C3: \quad & \sum_{j=0}^{M} I_{k,i,j} = 1, I_{k,i,j} \in \{0,1\} \end{aligned} \tag{17}$$

where I_i is the offloading decision set of UE_i. In $P1$, $C1$ is the execution delay constraint of the application generated by UE_i. $C2$ indicates the sequence of task execution the dependency between the tasks. $C3$ represents the decision of task offloading. We limit that a task can only be executed at one location at a time.

3. Proposed Algorithm

In this paper, we propose a Multi-population Coevolutionary Elite Genetic algorithm (MCE-GA) to solve this problem and make a reasonable offloading decision. We introduce multiple populations for parallel optimization. We endow different populations with different controls to achieve different search objectives. Each population evolves in parallel according to its own different evolutionary strategies and genetic manipulation. By co-evolution of migration operator, the optimal solution is obtained. Artificial selection operator is used to save the optimal individual and the convergence of the algorithm is judged in each evolution.

3.1. The Flow of MCE-GA

In this subsection, we introduce the flow of MCE-GA.

(1) Chromosome and Fitness Function

For simplicity, real number coding is used. In our algorithm, individuals are defined as chromosomes, indicating that each chromosome individual is a computation offloading decision to problem $P1$. We assume that UE_i has k tasks. The individual structure of chromosome is shown in Figure 4. Each chromosome individual contains the user's tasks scheduling policy. The resulting decision is variable for the task, i.e., choosing where the task offloading.

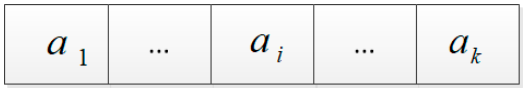


Figure 4. The chromosome structure.

We take the total overhead incurred by UE_i as the fitness value of the individual. Thus, the fitness function is defined as

$$Fitness = \beta * E_{i,total} + (1-\beta) * T_{i,total}, \tag{18}$$

where $E_{i,total}$ is the total energy of UE_i and $T_{i,total}$ is the total delay of UE_i.

(2) Initialization and Selection

We generate the initial populations randomly, but within the restraints in Problem $P1$. The individuals of the populations are generated and the variable values of genes are guaranteed to be within their range. Thus, we set the genes of the initial populations to $a_i(0) = random(M)$, where random(value) returns a random number in the interval

[0,value]. If $a_i == j$, then the task v_i is executed on MEC_j. We use $a_i = 0$ to represent the task v_i to be executed locally.

The selection operation is then performed within each population. For selection operations, two methods are used in a wide range, namely, roulette and tournaments. The roulette selection method calculates the probability of each individual according to the fitness value of the individual and randomly selects individuals to form the offspring population according to the probability. Therefore, the greater the fitness value of the individual, the greater the chance it has to be selected. Multiple rounds of selection are required to select mating individuals. A uniform random number between 0 and 1 is generated for each round, and this random number is used as the selection pointer to determine the selected individual. Due to random operation, the selection error of this selection method is also relatively large, and sometimes, the high fitness of the individual is not selected. In addition, the selection strategy roulette used in roulette is more suitable for the maximization problem. Therefore, a tournament selection strategy is adopted in this paper to select a certain number of individuals from the population at a time, and then select the best one to enter the offspring population. The first reason is that this method has a low computational cost. Another point is that choosing good parents is better than choosing only the best parents [28]. This method is more random and has a greater random error. It maintains diversity and has the chance to produce better individuals. Additionally, this method does not require the individual fitness value to be positive or negative.

Therefore, different populations take the selection operation independently. N individuals were randomly selected from the population for fitness comparison, and the highest fitness individuals were inherited to the next generation.

(3) Crossover and Mutation Operation

In order to preserve the diversity of the population and better solve the problem of offloading decision, MCE-GA performs iteration, crossover operation, and mutation operation on each population. Appropriate individuals will be selected for crossover and mutation operation. Then, offspring will be produced.

For crossover operation, we adopted a two-point crossover. Two-point crossover means that two intersection points are randomly set in the individual and then partial gene exchange is carried out. Figure 5 shows the example of the crossover operation for the computation offloading decision variables. First, two intersection points are randomly set in two individuals, and then, part of the chromosomes of two individuals between the two set intersections is exchanged.

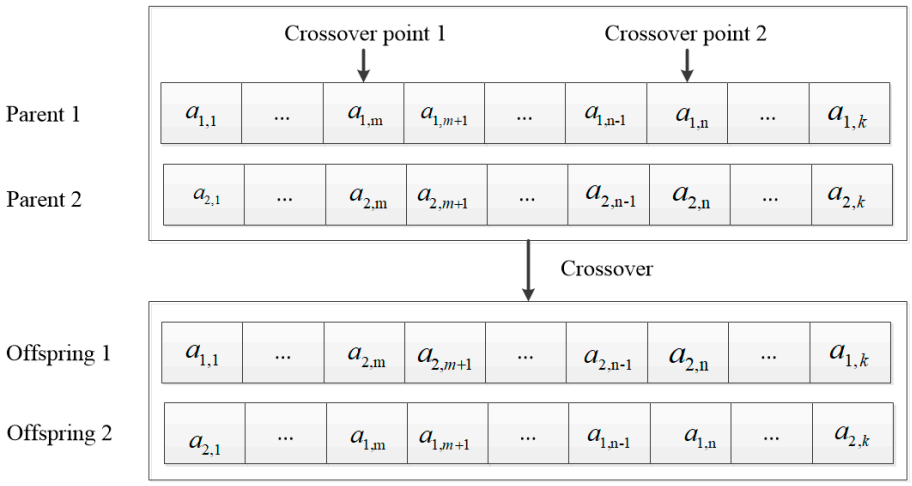


Figure 5. Crossover of the first row of chromosome.

Moreover, when setting the parameters of cross recombination, we generate the recombination probability list and assign different recombination probabilities to different

populations. We can maintain the diversity of the population better and avoid local optimality.

For mutation operation, a list of variation probabilities is generated and different variation probabilities are assigned to different populations. An individual is chosen and the mutation probability of each element in the chromosome is in the range of the variables.

In the case of mutation operation, when the variable value of the chromosome exceeds the range set by the individual chromosome variable, it is repaired. In this paper, truncation repair is adopted, i.e., the nearest boundary value of the element beyond the boundary range is taken.

(4) Migration

By setting different probability values and performing individual migration among populations according to the probability, the diversity of each population is maintained. All populations can share excellent genes, complete their own evolution, and finally, get the optimal solution. When making individual out, we choose preferred emigrated from individuals, and at the time of the move in individual, population with the method of choosing a bad replacement, so that we can speed up the convergence of the algorithm. In the Internet environment, most of the request to delay requirement is high, so offloading decisions need to be made quickly, so we use an elite reserved strategy.

(5) Elite Population Selection

The optimal individual of each evolutionary generation is preserved and compared with the optimal value of the previous generation, and the difference value is taken as the basis to judge the convergence of MCE-GA. For the elite population, we do not take crossover and mutation operation and only retain it.

3.2. Offloading Strategy Based on MCE-GA

In this subsection, we introduce our proposed MCE-GA algorithm in detail. The algorithm aims to solve resource allocation and task scheduling problem, as shown in Algorithm 1.

In Algorithm 1, first, we initialize multiple populations and some important parameters, which including evolutionary stagnation threshold, the maximum number of iterations, interval steps value of population migration, mutation operator probability, and crossover operator probability for each population. Moreover, we need to initialize a list to keep track of the elite population. Then, we calculate the fitness value (lines 3–5). The optimal individual in each population is updated to the elite population. The algorithm ends on the basis that the optimal individual in the elite population remains above the specified algebra. When the iteration does not end, the evolutionary operation continues. In the multi-population evolutionary operation, we assume that populations are independent of each other in selection, crossover, and mutation operation. So, for each population, we evolve individuals according to the selection, crossover, and mutation operations described above, and get a new generation of population (lines 8–14). We control population connections and coevolution through population migration. Therefore, the optimal individuals in a population are transferred to other populations through migration operations (lines 15–16). At each iteration, the elite population is renewed and the elite individuals of each population are retained (line 17). When evolution reaches the convergence standard, the value of the best individ-

ual is the user's offloading strategy. The fitness of the individual is the user's overhead.

Algorithm 1: MCE-GA

Inputs:
Population size list $PopNum$, evolutionary stagnation threshold ε, the iterations T, interval steps of migration $migFr$, mutation probability list p_m, crossover probability list p_c
Outputs:
Optimal offloading policy A, the total overhead C
1: Randomly initialize the populations Pop and Elite population $Elite\_pop$
2: Initialize the inputs
3: **For** $i = 1$ *to* $PopNum.size$ **do**
4: $Pop = Pop[i]$;
5: Evaluate the fitness value of each individual in the *i-th* Pop;
6: Update Elite population $Elite\_pop$;
7: **While** stopping criterion is not met **do**
8: **For** $i = 1$ *to* $PopNum.size$ **do**
9: offspring = Select ($Pop[i]$);
10: pop = Cross and Mutate (offspring);
11: $Pop[i] = Pop[i]$+ pop;
12: evaluate the fitness value of each individual in $Pop[i]$;
13: Select individuals to get a new generation of population;
14: **End For**
15: **IF** evolutionary algebra % $migFr == 0$ **do**
16: Carry out population migration;
17: Update Elite population $Elite\_pop$;
18: Return optimal offloading policy and the total overhead.
19: **End**

4. Simulation and Result

4.1. Simulation Setting

We consider an area of 500×500 m^2. We set the maximum latency of each UE up to 85% of the local latency. Referring to [24,29], we assume our simulation setting parameters. We list the important simulation parameters in Table 2.

Table 2. The simulation parameters.

Simulation Parameters	Value
Channel bandwidth	180 kHz
Path loss exponent	3
Background noise	10^{-13}
Number of tasks	61,223
Data size of tasks	300~1000 kb
Transmission power of UE	3 W
Computation capacity of MEC server	5 GHz
Computation capacity of UE	0.5–1 GHz
Channel fading coefficient	10^{-3}
Tradeoff parameter β	0.5
Processing density of UE	500~800 cycle/bit
Distance between UE and MEC server	80~100 m
Distance of MEC servers	50~100 m
Coefficient factor of device's chip architecture	10^{-20}
Number of MECs	3–7
Ratio of the output data size to the input data size	0.001~0.005
Interval steps of population migration	5
Size of the populations	10,152,025

In this paper, we consider the three baseline algorithms to evaluate the performance, and make a comparative analysis of their experimental results.

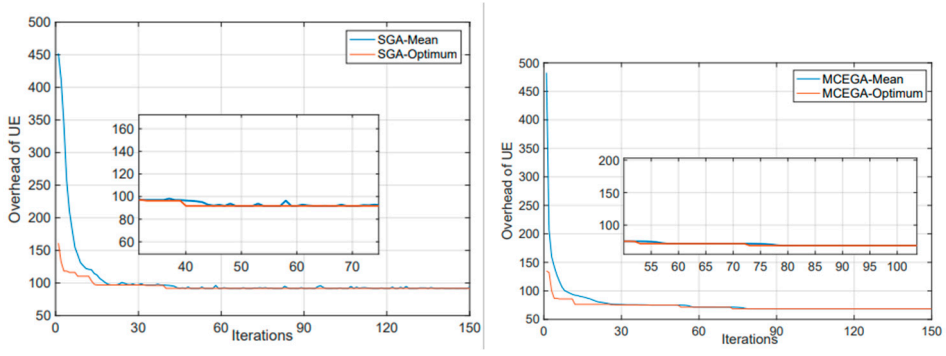
Random selection algorithm (Random): The tasks of applications randomly select offloading locations.

Greedy algorithm (Greedy): The tasks of applications select the best offloading locations in the current situation. Greedy is to achieve global optimization through local optimization, and to construct the optimal solution step by step. At each stage, a seemingly optimal decision is made. Once the decision is made, it will not be changed. Therefore, with the greedy algorithm, the current task of application always chooses what looks like the best offloading location based on its predecessor task processing and the local user and MEC server situation, but ignores its successor tasks. Standard genetic algorithm (SGA): SGA introduces the biological evolutionary principle of "survival of the fittest, survival of the fittest" into the coded tandem population formed by optimized parameters. SGA starts with an initial set of random offloading strategy and optimizes the offloading strategy through some standard genetic operations such as selection operation, crossover operation, and mutation operation until reaching a better offloading strategy or convergence. Because the optimization does not depend on the gradient, it has strong robustness and global search ability. However, immature convergence is indeed a phenomenon that cannot be ignored in the standard genetic algorithm. It mainly shows that all individuals are in the same state and stop evolving. By comparing the proposed MCE-GA algorithm with these three algorithms, the effectiveness of MCE-GA is verified.

4.2. Convergence Analysis

In this subsection, the convergence of SGA and MCE-GA are analyzed. The convergence of genetic algorithms has been demonstrated in literature [30,31].

Figure 6 shows the convergence situation of MCE-GA and SGA. We consider that the MEC servers are lack of computing resource and mobile terminals have low latency requirements, so the iteration is set to 150 times in this work. The black line shows the average fitness value of the population, and red line shows the optimal fitness value of the population. In this paper, the fitness value represents the overhead of the UE when computing tasks, so the lower the fitness value, the lower the overhead, and the better the strategy. We can see that MCE-GA basically reaches convergence and maintains stability quickly, and the average fitness value is basically consistent with the best fitness value. However, the overhead of SGA still fluctuates in a small range which has not reached convergence.



(**a**) (**b**)

Figure 6. The convergence situation of MCEGA (**b**) and Standard genetic algorithm (SGA) (**a**).

Figure 7 shows the different convergence of SGA and MCE-GA. We can see that SGA is easy to fall into "premature" and converge to the local optimal solution. However, MCE-GA algorithm introduces multiple population optimization search at the same time. To achieve the purpose of different searches, different populations are endowed with different control parameters. The introduction of migration operator is to connect multiple populations

and realize the co-evolution, which makes MCE-GA obtain the optimal solution. Thus, the overhead of MCE-GA is better than SGA.

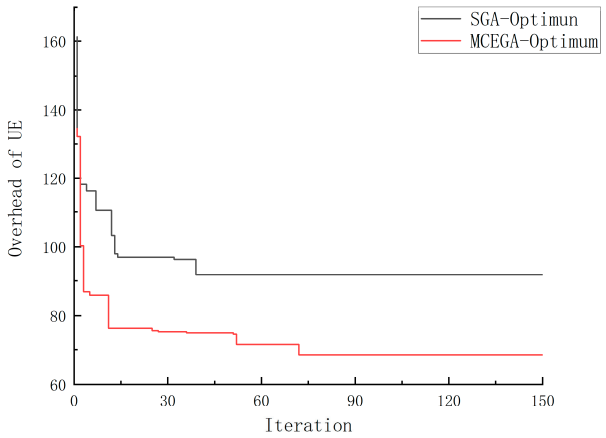


Figure 7. Convergence comparison between MCE-GA and SGA.

4.3. Performance Analysis

In this subsection, we analyze the performance of MCE-GA.

Figure 8 shows the impact of different number settings of MEC server on device overhead. The abscissa is the number of MEC servers, and the ordinate is the device overhead. The number of MEC servers is set at 3–7. As the number of MEC servers increases, the overhead of application generated by UE decreases. This is because, the more the MEC servers there are, the more the rich computing resources there are in the system; so, users can greatly reduce the delay in executing tasks, thus reducing the overall overhead. However, we find that when the number of MECs grows from 5 to 7, the overhead is reduced but not by much. This is because, when the number of MECs increases to a certain extent, MEC server resources is particularly rich, where the offloading location is not significantly affected in this case. Therefore, how to set the number of MEC servers to achieve optimal system performance is particularly important. We set the number of MECs to 7, and by comparing the four algorithms, we can see that MCE-GA algorithm can better reduce the overhead of devices. Compared with random method, greedy method, and standard genetic algorithm, the optimization rate of MCE-GA method is 55.5%, 49%, and 7.1%, respectively. Since the random method is to randomly select the unloading location, the overhead incurred is random and expensive. The greedy strategy is to select the local optimal unloading location for the current subtask, but not to consider the offloading location globally, so the overhead is large. The standard genetic algorithm and MCEGA algorithm take into account the global offloading strategy, so that the overhead is small, but the standard genetic algorithm is easy to fall into premature. The offloading decision made by MCE-GA is better. MCE-GA can better reduce the overhead.

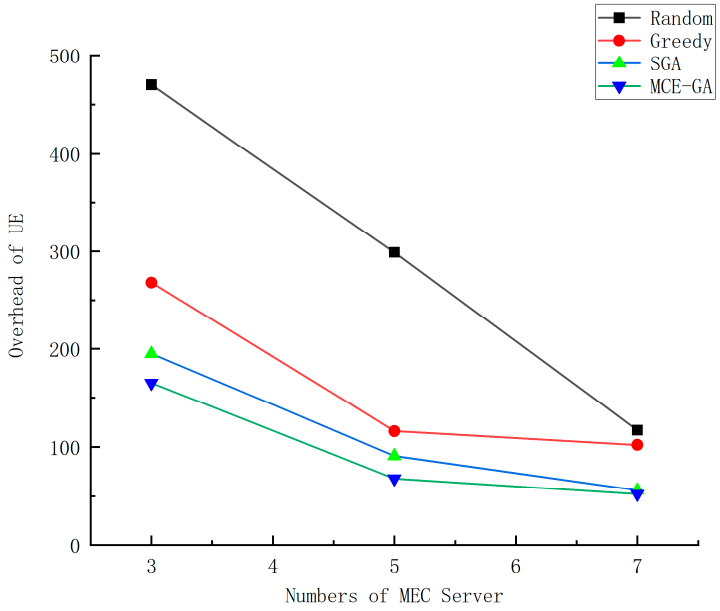


Figure 8. The execution overhead of different number of MEC server.

Figure 9 shows the overhead of UE with different offloading strategies at different data sizes. From Figure 8, we find that it is better to set the number of servers to 5, because when the number of servers increases to 7, the reduction in overhead is not obvious, but the cost of adding MEC servers increases. Therefore, number of MEC servers is set to 5. Then, we set the data size generated by each task, at 300–1000 KB. The experimental result figure shows that with the increase in data volume, the total overhead of the system will increase. This is because the increase in data volume will consume more computing resources of UE and MEC servers, resulting in an increase in the overall system overhead. When the amount of data is 300 KB, we find that there is little difference between the greedy strategy, the standard genetic algorithm, and the MCE-GA algorithm. Because the amount of data that needs to be computed is small and the computing resources in the system is rich, the overall cost does not differ much. However, the random strategy is a little big; because the task dependence itself is a very complex problem, the improper selection of the strategy will cause a lot of ready delay, which will lead to extremely high overhead. As the data size increases, the advantages of MCE-GA algorithm are reflected, because when the data volume is larger, the resources of MEC server are limited, the choice of offloading strategy becomes particularly important. A good strategy can make the system produce lower overhead. Compared with the random method, greedy method, and standard genetic algorithm, the optimization rate of MCE-GA method was 77.2%, 41.3%, and 25.2%, respectively, when the data volume was 500 KB. The offloading decision made by MCE-GA is better.

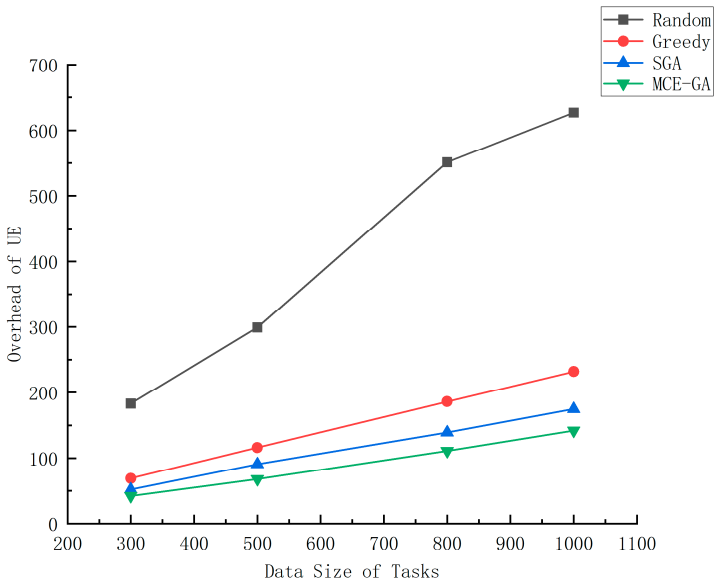


Figure 9. The execution overhead of different data size of tasks.

Figure 10 shows the total device overhead when multiple UEs perform task computation. We set the number of devices from 10 to 50 randomly. The number of tasks generated by UEs is random, and the data size of each task is random. We can find that as the number of UEs increases, the overall system overhead is on the rise. Additionally, MCE-GA has more advantages. As the more UEs there are, the more computing resources system need. The computing resources of MEC servers and UEs are not infinite, so how to allocate resources and how to offload intelligent devices are particularly important. In terms of offloading strategy selection, MCE-GA has a greater advantage. When the number of equipment is 20, we analyze the performance of the algorithm. Compared with random method, greedy method, and standard genetic algorithm, the optimization rate is 72.4%, 38.6%, and 19.3%, respectively. The proposed algorithm in this paper can make a better offloading decision and reduce overhead of UEs.

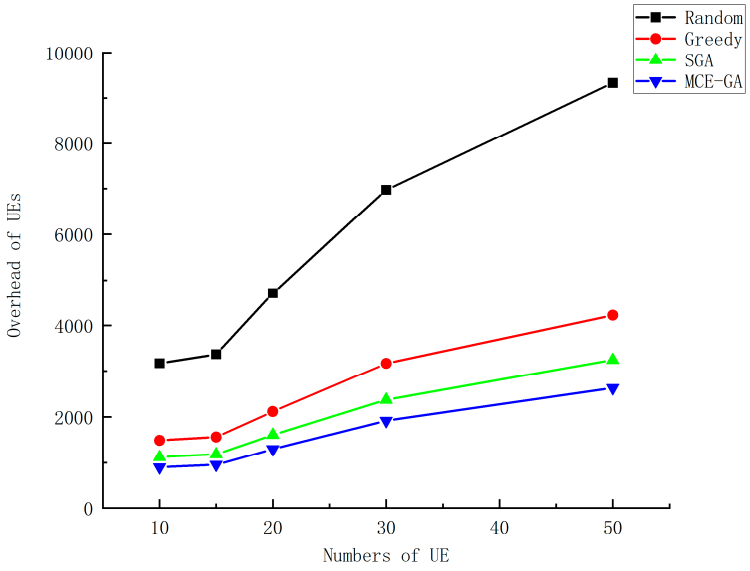


Figure 10. The execution overhead of different numbers of device.

5. Conclusions

In this paper, we propose the problem of task partitioning computation offloading strategy under multi-UE and multi-MEC model with limited resources. First, we consider fine-grained task computation offloading of fine-grained tasks, which have dependencies, model the user-generated mobile application as a directed acyclic graph, and make the parallel processing of tasks possible. Second, in order to meet low energy consumption and low latency service requirements, we consider the execution delay and energy consumption of applications generated by UEs, and formulate a problem of minimizing overhead of UEs. Then, we propose MCE-GA algorithm for joint optimization. Finally, we study the convergence and performance of MCE-GA by simulation. Simulation results show that the proposed algorithm can find a reasonable allocation of resources and reduce the overhead of the whole system. It is superior to Random, Greedy, and SGA in reducing overhead of UEs.

For future work, we will take the cloud into consideration and consider the architecture of edge-cloud collaboration. Moreover, we will continue our research by considering user mobility and dynamic computation offloading. Mobility is an inherent feature of many emerging applications, such as AR to assist museum visits to enhance the visitor experience. Therefore, mobility management is a key problem that needs to be solved urgently in computing offloading, and it is also one of the major challenges to further design efficient computing offloading schemes.

Author Contributions: Conceptualization, J.F., S.L. and J.S.; methodology, J.F., J.S., S.L. and M.Z.; validation, J.F., J.S., S.L. and Z.Y.; formal analysis, J.F.; investigation, J.F., J.S. and S.L.; resources, J.F. and S.L.; writing—original draft preparation, J.S.; writing—review and editing, J.F., J.S. and S.L.; supervision, J.F., S.L., S.L., M.Z. and Z.Y.; project administration, J.F.; funding acquisition, J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Beijing Natural Science Foundation, grant number No. 4192007, and the National Natural Science Foundation of China, grant number No. 61202076.

Acknowledgments: This work is supported by the Beijing Natural Science Foundation (4192007), and the National Natural Science Foundation of China (61202076), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ullah, R.; Rehman, M.A.U.; Naeem, M.A.; Kim, B.-S.; Mastorakis, S. ICN with edge for 5G: Exploiting in-network caching in ICN-based edge computing for 5G networks. *Futur. Gener. Comput. Syst.* **2020**, *111*, 159–174. [CrossRef]
2. Ericsson. *Cellular Networks for Massive IoT—Enabling Low Power Wide Area Applications*; Ericsson: Stockholm, Sweden, 2016; pp. 1–13.
3. Barbarossa, S.; Ceci, E.; Merluzzi, M.; Calvanese-Strinati, E. Enabling effective Mobile Edge Computing using millimeterwave links. In Proceedings of the 2017 IEEE International Conference on Communications Workshops (ICC Workshops), Paris, France, 21–25 May 2017; pp. 367–372.
4. Belli, D.; Chessa, S.; Foschini, L.; Girolami, M. A Social-Based Approach to Mobile Edge Computing. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 00292–00297. [CrossRef]
5. Roman, R.; Lopez, J.; Mambo, M. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. *Futur. Gener. Comput. Syst.* **2018**, *78*, 680–698. [CrossRef]
6. Wu, Q.; Chen, X.; Zhou, Z.; Chen, L. Mobile Social Data Learning for User-Centric Location Prediction with Application in Mobile Edge Service Migration. *IEEE IoT J.* **2019**, *6*, 7737–7747. [CrossRef]
7. Yangchen, C.; Guosheng, Z.; Xiaoyun, Q.; Jie, Z. Research on cloud vr based on 5g edge computing. *Inf. Commun.* **2019**, *33*, 1–3.
8. Li, Y.; Frangoudis, P.A.; Hadjadj-Aoul, Y.; Bertin, P. A mobile edge computing-based architecture for im-proved adaptive HTTP video delivery. In Proceedings of the 2016 IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, Germany, 31 October–2 November 2016; pp. 1–6.

9. Li, L.; Li, Y.; Hou, R. A Novel Mobile Edge Computing-Based Architecture for Future Cellular Vehicular Networks. In Proceedings of the 2017 IEEE Wireless Communications and Networking Conference (WCNC), San Francisco, CA, USA, 19–22 March 2017; pp. 1–6.
10. Datta, S.K.; Bonnet, C.; Haerri, J. Fog Computing architecture to enable consumer centric Internet of Things services. In Proceedings of the 2015 International Symposium on Consumer Electronics (ISCE), Madrid, Spain, 24–26 June 2015; pp. 1–2.
11. Qiu, X.; Liu, L.; Chen, W.; Hong, Z.; Zheng, Z. Online Deep Reinforcement Learning for Computation Offloading in Blockchain-Empowered Mobile Edge Computing. *IEEE Trans. Veh. Technol.* **2019**, *68*, 8050–8062. [CrossRef]
12. Luong, N.C.; Xiong, Z.; Wang, P.; Niyato, D. Optimal Auction for Edge Computing Resource Management in Mobile Blockchain Networks: A Deep Learning Approach. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6.
13. Kang, J.; Yu, R.; Huang, X.; Wu, M.; Maharjan, S.; Xie, S.; Zhang, Y. Blockchain for Secure and Efficient Data Sharing in Vehicular Edge Computing and Networks. *IEEE IoT J.* **2018**, *6*, 4660–4670. [CrossRef]
14. Ullah, R.; Rehman, M.A.U.; Kim, B.-S. Design and Implementation of an Open Source Framework and Prototype For Named Data Networking-Based Edge Cloud Computing System. *IEEE Access* **2019**, *7*, 57741–57759. [CrossRef]
15. Sonkoly, B.; Haja, D.; Németh, B.; Szalay, M.; Czentye, J.; Szabó, R.; Ullah, R.; Kim, B.-S.; Toka, L. Scalable edge cloud platforms for IoT services. *J. Netw. Comput. Appl.* **2020**, *170*, 102785. [CrossRef]
16. Yang, L.; Zhang, H.; Li, M.; Guo, J.; Ji, H. Mobile Edge Computing Empowered Energy Efficient Task Offloading in 5G. *IEEE Trans. Veh. Technol.* **2018**, *67*, 6398–6409. [CrossRef]
17. Zhang, J.; Hu, X.; Ning, Z.; Ngai, E.C.-H.; Zhou, L.; Wei, J.; Cheng, J.; Hu, B. Energy-Latency Tradeoff for Energy-Aware Offloading in Mobile Edge Computing Networks. *IEEE IoT J.* **2017**, *5*, 2633–2645. [CrossRef]
18. Tong, Z.; Deng, X.; Ye, F.; Basodi, S.; Xiao, X.; Pan, Y. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Inf. Sci.* **2020**, *537*, 116–131. [CrossRef]
19. Zhang, H.; Guo, J.; Yang, L.; Li, X.; Ji, H. Computation offloading considering fronthaul and backhaul in small-cell networks integrated with MEC. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017; pp. 115–120.
20. Xu, J.; Li, X.; Ding, R.; Liu, X. Energy efficient multi-resource computation offloading strategy in mobile edge com-puting. *CIMS* **2019**, *25*, 954–961.
21. Xing, Z.; Jianhua, P.; Wei, Y. A Privacy-aware Computation Offloading Method Based on Lyapunov Optimization. *J. Electron. Inf. Technol.* **2020**, *42*, 704–711.
22. Liu, L.; Liu, X.; Zeng, S.; Wang, T.; Pang, R. Research on virtual machines migration strategy based on mobile user mobility in mobile edge computing. *J. Chongqing Univ. Posts Telecommun.* **2019**, *31*, 158–165.
23. Ding, Y.; Liu, C.; Zhou, X.; Liu, Z.; Tang, Z. A Code-Oriented Partitioning Computation Offloading Strategy for Multiple Users and Multiple Mobile Edge Computing Servers. *IEEE Trans. Ind. Inform.* **2019**, *16*, 4800–4810. [CrossRef]
24. Qiuping, L.; Junhui, Z.; Yi, G. Computation offloading and resource management scheme in mobile edge computing. *Telecommun. Sci.* **2019**, *35*, 36.
25. Sklar, B. Rayleigh fading channels in mobile digital communication systems. I. Characterization. *IEEE Commun. Mag.* **1997**, *35*, 136–146. [CrossRef]
26. Zhang, W.; Wen, Y.; Guan, K.; Kilper, D.; Luo, H.; Wu, D.O. Energy-optimal mobile cloud computing under sto-chastic wireless channel. *IEEE Trans. Wirel. Commun.* **2013**, *12*, 4569–4581. [CrossRef]
27. Wallenius, J.; Dyer, J.S.; Fishburn, P.C.; Steuer, R.E.; Zionts, S.; Deb, K. Multiple criteria decision making, multiat-tribute utility theory: Recent accomplishments and what lies ahead. *Manag. Sci.* **2008**, *54*, 1336–1349. [CrossRef]
28. Lazar, E.; Petreus, D.; Etz, R.; Patarau, T. Minimization of operational cost for an Islanded Microgrid using a real coded Genetic Algorithm and a Mixed Integer linear Programming method. In Proceedings of the 2017 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM) & 2017 Intl Aegean Conference on Electrical Machines and Power Electronics (ACEMP), Fundata, Romania, 25–27 May 2017; pp. 693–698.
29. Guo, F.; Zhang, H.; Ji, H.; Li, X.; Leung, V.C.M. An Efficient Computation Offloading Management Scheme in the Densely Deployed Small Cell Networks with Mobile Edge Computing. *IEEE/ACM Trans. Netw.* **2018**, *26*, 2651–2664. [CrossRef]
30. Rudolph, G. Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Netw.* **1994**, *5*, 96–101. [CrossRef] [PubMed]
31. Bhandari, D.; Murthy, C.A.; Pal, S.K. Genetic algorithm with elitist model and its convergence. *Int. J. Pattern Recognit. Artif. Intell.* **1996**, *10*, 731–747. [CrossRef]