A Multiobjective Computation Offloading Algorithm for Mobile Edge Computing

Fuhong Song, Huanlai Xing, Shouxi Luo, Dawei Zhan, Penglin Dai, Rong Qu

***Abstract*—In mobile edge computing (MEC), smart mobile devices (SMDs) with limited computation resources and battery lifetime can offload their computing-intensive tasks to MEC servers, thus to enhance the computing capability and reduce the energy consumption of SMDs. Nevertheless, offloading tasks to the edge incurs additional transmission time and thus higher execution delay. This paper studies the trade-off between the completion time of applications and the energy consumption of SMDs in MEC networks. The problem is formulated as a multiobjective computation offloading problem (MCOP), where the task precedence, i.e. ordering of tasks in SMD applications, is introduced as a new constraint in the MCOP. An improved multiobjective evolutionary algorithm based on decomposition (MOEA/D) with two performance enhancing schemes is proposed. 1) The problem-specific population initialization scheme uses a latency-based execution location initialization method to initialize the execution location (i.e. either local SMD or MEC server) for each task. 2) The dynamic voltage and frequency scaling based energy conservation scheme helps to decrease the energy consumption without increasing the completion time of applications. The simulation results clearly demonstrate that the proposed algorithm outperforms a number of state-of-the-art heuristics and meta-heuristics in terms of the convergence and diversity of the obtained nondominated solutions.**

***Index Terms*—Computation offloading, dynamic voltage and frequency scaling, mobile edge computing, multiobjective evolutionary algorithm.**

I. Introduction

WITH the rapid development of mobile communication technologies, smart mobile devices (SMDs) including smartphones, tablets, laptops and smartwatches have become the main platforms to support various mobile applications such as banking, education, healthcare, travel, business, games, face recognition, augmented reality, and natural language processing, etc. [1][2]. Nowadays, although SMDs are becoming increasingly powerful in terms of computing capability, they are, however, still not able to support computing-intensive applications. On the one hand, SMDs with limited computing capability may cause high latency, thus failing to meet the required quality of service (QoS) demand. On the other hand, high battery consumption by computing-intensive applications may also significantly degrade the quality of experience (QoE) for end users.

With more computing and storage resources in cloud servers, mobile cloud computing (MCC) [3] has been envisioned as a potential solution to deal with the above mentioned problems. MCC migrates computational tasks to cloud servers, thus to reduce the computational burden and energy consumption of local SMDs. This is referred to as the computation offloading problem (COP). Nevertheless, cloud servers are usually geographically faraway from SMDs, resulting into high transmission delay and low response speed. Obviously, MCC is not suitable for scenarios involving delay-sensitive applications, as QoE cannot be properly guaranteed. Actually, the computation offloading in MCC is only suitable for delay-tolerant and computation-intensive applications, such as online social networks, mobile e-commerce, remote learning, etc. On the other hand, MEC relocates cloud computing resources to the edge of networks in close proximity to SMDs, ensuring lower end-to-end delay and faster response [4][5][6]. The computation offloading in MEC is more appropriate for supporting delay-sensitive and computation-intensive applications, such as virtual reality, autonomous driving, and interactive online games and so on. With the demand for delay-sensitive applications ever increasing, it is hence more practical to study the COP problem in MEC.

In general, MEC servers are lightweight regarding the computing capability, because their economic and scalable deployment should be considered. It is thus not feasible to offload all computational tasks from SMDs to the MEC severs. More data transmission over communication channels also leads to higher transmission delay. To avoid overloading, SMDs should offload appropriate amount of computational tasks to MEC servers, which also helps to reduce the battery consumption of SMDs.

In MEC, the completion time of applications and the energy consumption of SMDs conflict with each other. In other words, improving one of them would deteriorate the other. The computational problem of reasonably offloading tasks between SMDs and MEC servers, i.e. COP, has become one of the most challenging research topics in the area of MEC.

Manuscript received December 7, 2019; This work was supported in part by China Postdoctoral Science Foundation (2019M663552), National Natural Science Foundation of China (No. 61802319), the Fundamental Research Funds for the Central Universities, China Scholarship Council, P. R. China, and University of Nottingham, United Kingdom. *(Corresponding author: Huanlai Xing.)*

F. Song, H. Xing, S. Luo, D. Zhan, and P. Dai are with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China (e-mail: fhs@my.swjtu.edu.cn; hxx@home.swjtu.edu.cn; sxluo@swjtu.edu.cn; zhandawei@swjtu.edu.cn; penglindai@swjtu.edu.cn).

R. Qu is with the School of Computer Science, University of Nottingham, Nottingham NG8 1BB, United Kingdom (e-mail: rong.qu@nottingham.ac.uk).

In this paper, we model the computation offloading in MEC as a multiobjective computation offloading problem (MCOP). A multiobjective evolutionary algorithm based on decomposition (MOEA/D) is adopted for solving it [7].

The main contributions are summarized as follows:

1) A MCOP problem in MEC environment is modeled, where the average completion time of applications and the average energy consumption of SMDs are defined as two objectives. On each SMD, only one application, with an ordered list of tasks, runs at a time. To our knowledge, this is the first model in MEC considering the task-precedence constraints within each application in MCOP.
2) An improved MOEA/D algorithm with two performance-enhancing schemes, namely MOEA/D-MCOP, is proposed. The first scheme, a problem-specific population initialization scheme generates a set of high-quality solutions to MCOP, where a latency-based execution location initialization (LELI) method is designed to determine the initial execution location (i.e. local SMD or MEC server) for each task, guiding the exploration towards promising regions in the search space. The second scheme, a dynamic voltage and frequency scaling based energy conservation scheme, aims at reducing the energy consumption of SMDs.
3) For the new MCOP built in this paper, there exists no benchmark instance in the literature. A set of test instances are thus generated to verify the performance of the proposed MOEA/D-MCOP, and presented to the research community for further investigations on this emerging topic. The simulation results clearly demonstrate that the proposed algorithm obtains high-quality nondominated solutions and outperforms a number of state-of-the-art MOEAs and heuristic algorithms against several evaluation criteria.

The remainder of this paper is organized as follows. The related work is introduced in Section II. In Section III, we present the MEC system model and formulate the bi-objective MCOP problem. Section IV briefly reviews the multiobjective optimization problem and the original MOEA/D. The proposed MOEA/D-MCOP is explained in Section V. The simulations and performance analysis are discussed in Section VI. Section VII presents the conclusions and future work.

II. Related Work

The COP problem has received an increasing research attention from both academia and industry [8]. In general, completion time and energy consumption are considered as typical criteria for COP performance evaluation, i.e. as objectives of minimizing the completion time, minimizing the energy consumption, and minimizing both of them at the same time.

When a SMD offloads computing-intensive tasks to a MEC server, the completion time is one of the important criteria for QoE evaluation. Liu *et al*. [9] adopted Markov decision process to determine execution locations for tasks. A transmission policy was devised based on the queueing state of task buffer, the transmission unit state and the local processing unit state. The average completion time of tasks was minimized by an efficient one-dimensional search algorithm. Mao *et al*. [10] developed a green MEC system with energy harvesting and proposed a low complexity online algorithm, i.e. Lyapunov optimization based dynamic computation offloading algorithm (LODCO) to reduce the execution latency by jointly determining the offloading decision, the CPU-cycle frequency and the transmission power. Yang *et al*. [11] investigated the scheduling problem of multi-user computing partitioning and cloud resource computing offloading. The average completion time of multiple users, rather than a single user, was minimized by an offline heuristic algorithm. Dinh *et al*. [12] took both fixed and elastic CPU frequencies of SMDs into account. A semidefinite relaxation (SDR) based approach was proposed to minimize the execution time of all tasks.

Energy consumption of SMDs is also a main concern in COP. Muñoz *et al*. [13] proposed a framework to jointly optimize the usage of computational and radio resources, where multiple antennas were used in SMDs and femto access points. The energy consumption was minimized by optimizing the communication time and the amount of data offloaded to a femto access point. Tong *et al*. [14] aimed at obtaining a trade-off between energy consumption of SMDs and QoS of applications. An application-aware wireless transmission scheduling algorithm was presented to minimize the energy consumption, subject to the application deadline. Masoudi *et al*. [15] considered three practical constraints, i.e. the backhaul capacity, the maximum tolerable delay, and the interference level. They proposed a joint power allocation and decision-making algorithm to minimize the power consumption of SMDs. Wang *et al*. [16] presented an integrated framework for computation offloading and interference management, where the physical resource block, the computation offloading decision and the computation resource allocation were taken into consideration for reducing energy consumption. Mahmoodi *et al*. [17] modeled the COP as a linear optimization problem on energy consumption, where the communication delay, the overall application execution time and the component precedence ordering were taken into account. Xu *et al*. [18] proposed an energy-aware computation offloading scheme, where simple additive weighting and multiple criteria decision marking were used to determine an optimal solution. In [19], an energy-efficient COP problem in 5G MEC was investigated, considering fronthaul and backhaul links. The overall energy consumption was minimized by an artificial fish swarm algorithm, subject to the completion time demand. In [20], a security and energy efficient computation offloading scheme based on genetic algorithm was presented. Guo *et al*. [21] formulated a cloud-MEC collaborative computation offloading problem. The authors presented an approximation collaborative computation offloading scheme to minimize the energy consumption of all mobile devices. Zhang *et al*. [22] proposed a collaborative task execution scheduling algorithm to solve the delay-constrained workflow scheduling problem in MCC. The energy consumption of SMDs was minimized, with the application delay deadline satisfied. Guo *et al*. [23] studied an energy-efficient computation offloading management scheme in MEC with small cell networks. A hierarchical GA and PSO-based computation algorithm was developed to minimize the energy consumption of all mobile devices. Kuang *et al*. [24] formulated a multi-user offloading game problem in the OFDMA communication system. The authors presented an offloading game mechanism to maximize the number of energy-saving devices, including a beneficial offloading threshold algorithm and a beneficial offloading group algorithm. It minimized the energy cost while considering the application's deadline and risk probability. Lin *et al*. [25] applied dynamic voltage and frequency scaling (DVFS) to minimize SMD energy consumption in MCC environment, where task precedence requirements within any application were satisfied. However, the authors assumed that there was a single SMD in their MCC system, which was impractical. In real world applications, multiple SMDs are active at the same time, and some of them may offload their computation tasks to cloud.

On the one hand, a smaller completion time requires more tasks to be executed on local SMDs, which leads to higher battery consumption. On the other hand, to keep the battery consumption at a lower level requires more computations to be offloaded to the edge. Some researchers hence treated the completion time of applications and the energy consumption of SMDs as equally important, i.e. minimizing them simultaneously. Zhang *et al*. [26] considered single and multicell MEC network scenarios, and proposed an integrated framework for computation offloading and resource allocation. An iterative search algorithm was developed to strike a balance between execution time and energy consumption. Peng *et al*. [27] developed an optimal task scheduling scheme for SMDs using the DVFS technology and the whale optimization algorithm. Considering the operating CPU-cycle frequency, the task execution position and sequence, this scheme could optimize both of the objectives simultaneously. Guo *et al*. [28] studied energy-efficient COP subject to the application execution latency. An energy-efficient dynamic offloading and resource scheduling (eDors) scheme was proposed to reduce the execution latency and the energy consumption. Wang *et al*. [29] modeled an energy-efficient M/M/n-based COP with both of the objectives. A distributed algorithm considering transmission power allocation, strategy selection and clock frequency control was proposed. Cui *et al*. [30] investigated the tradeoff between the completion time and the energy consumption subject to end user requirements, and presented an improved fast and elitist nondominated sorting genetic algorithm (NSGA-II).

In summary, considering the completion time and energy consumption as two objectives represents one of the main streams in the current research on MEC computation offloading. To the best of our knowledge, however, task precedence has not been considered in the existing MCOPs. This presents a practical constraint in many applications. For example, in any face recognition system, object detection cannot be launched before the completion of video/image collection. This motivates us to model a new MCOP with a realistic task precedence constraint.

Most of the existing algorithms for MCOP evaluate solutions using weighted sum of multiple objectives. The fact that the objectives conflict with each other in MCOP has been omitted, thus a single solution cannot be optimized against all objectives. In research, NSGA-II has thus been employed to solve MCOP [30]. As a multiobjective evolutionary algorithm (MOEA), NSGA-II has unveiled promising advantage, i.e. providing a set of nondominated solutions for decision making in a single run. Nevertheless, as we observe in this paper, NSGA-II not only is likely to be stuck into local optima, but also converges slowly. MOEA/D decomposes a multiobjective optimization problem (MOP) into a number of scalar optimization subproblems and solves each of them at the same time. It has been reported that MOEA/D achieves better optimization performance with lower computational overhead, compared with NSGA-II [31][32][33]. This motivates us to investigate MOEA/D to address the newly modeled MCOP in this paper.

III. System Model and Problem Formulation

A. System Overview

A MEC system consists of one macro eNodeB node (MeNB) and a set of small eNodeB nodes (SeNBs) [26], as shown in Fig. 1. MeNB is equipped with a MEC server capable of executing multiple computing-intensive tasks in parallel. The MEC server can dynamically allocate its computing resources to execute tasks offloaded from different SMDs. All SeNBs are connected to MeNB via wired lines. Each SeNB forms a small cell, connecting to a set of SMDs via wireless channels.

Each task in an application can be run either on local SMDs or the MEC server. Computation offloading incurs when some tasks are offloaded from SMDs to the MEC server, and the data delivery relies on relay of SeNB and MeNB.

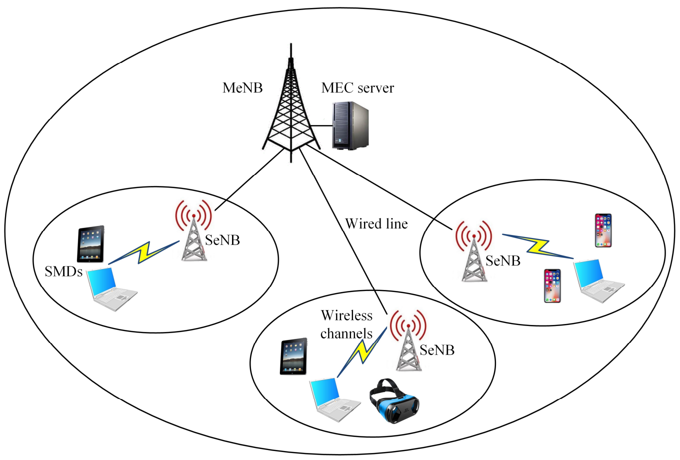


Fig. 1. An example MEC system.

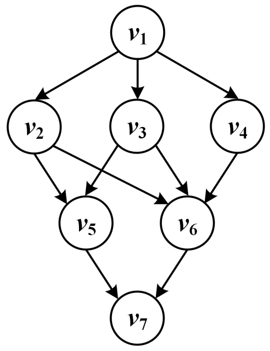


Fig. 2. An example DAG of an application.

There are two commonly used structural representation methods to denote an application, namely the graph-based method [34] and the language-based method [35]. In particular, the graph-based method includes directed acyclic graph **(**DAG) [20][25][27][28][36] and Petri Net [37]. DAG based model is one of the most popular methods. Therefore, each application on a SMD is modeled as a DAG task structure. Fig. 2 shows an example DAG of an application.

In this paper, time sharing is adopted in the MEC system and the minimum time unit is referred to as time interval (e.g. several seconds). We assume that in any time interval, for any SMD, there is only one application being executed. However, any SMD is allowed to run different applications in different time intervals, enabling co-existence of multiple applications.

B. System Model

For the MeNB, its associated MEC server is with a computing capability of F. Denote the i-th SeNB as π_i, $i = 1,\ldots,S$, where S is the number of SeNBs. Let N_i^{SMD} and $U_{i,j}$ be the number of SMDs and the j-th SMD in the i-th small cell associated with π_i, respectively.

Denote the application to be executed on SMD $U_{i,j}$ by a DAG $G_{i,j} = (V_{i,j}, E_{i,j})$, where $V_{i,j}$ and $E_{i,j}$ are the task and precedence constraint sets, respectively, $i = 1,\ldots,S$, and $j = 1,\ldots, N_i^{SMD}$. An *application* is also referred to as a *task graph* $G_{i,j}$, which is composed of $N_{i,j}$ tasks, where $N_{i,j} = |V_{i,j}|$. Let $v_{i,j,k} \in V_{i,j}$ be the k-th task in task set $V_{i,j}$, $k = 1,\ldots,N_{i,j}$. Edge $e(v_{i,j,k}, v_{i,j,l}) \in E_{i,j}$ defines the task-precedence constraint from task $v_{i,j,k}$ to task $v_{i,j,l}$, meaning $v_{i,j,l}$ cannot be executed until $v_{i,j,k}$ is completed.

Let $pre(v_{i,j,k})$ and $suc(v_{i,j,k})$ be the sets of the immediate predecessors and successors of task $v_{i,j,k}$, respectively. For a task graph $G_{i,j}$, denote the *start* and *end* tasks by $v_{i,j,start}$ and $v_{i,j,end}$, respectively. Taking the task graph in Fig. 2 as an example, for task v_6, its associated sets of immediate predecessors and successors are $pre(v_6) = \{v_2, v_3, v_4\}$ and $suc(v_6) = \{v_7\}$; v_1 and v_7 are the start and end tasks, respectively.

Each task $v_{i,j,k}$ is modeled as a 3-tuple set $v_{i,j,k} = (c_{i,j,k}, d_{i,j,k}, o_{i,j,k})$, where $c_{i,j,k}$ is the number of CPU cycles required to perform $v_{i,j,k}$, and $d_{i,j,k}$ and $o_{i,j,k}$ are the input and output data sizes of $v_{i,j,k}$, respectively. The input data of $v_{i,j,k}$ includes the input parameters, the program code, and the output data generated by all its immediate predecessors in $pre(v_{i,j,k})$. The main notations used in this paper are summarized in Table I.

TABLE I
SUMMARY OF THE MAIN NOTATIONS

Notation	Definition
B_{total}	Total bandwidth offered by the MEC system
$c_{i,j,k}$	Number of CPU cycles required to perform task $v_{i,j,k}$
$d_{i,j,k}$	Input data size of task $v_{i,j,k}$
$e(v_{i,j,k}, v_{i,j,l})$	Task-precedence constraint from task $v_{i,j,k}$ to task $v_{i,j,l}$
$E_{i,j}$	Set of the task precedence constraints of application $G_{i,j}$
F	Computing capability of the MEC server
$f_{i,j,h}^{actual}$	Actual computing frequency on the h-th core of SMD $U_{i,j}$
$f_{i,j,h}^{max}$	Maximum computing frequency on the h-th core of SMD $U_{i,j}$
$G_{i,j}$	Application (task graph) to be executed on SMD $U_{i,j}$
$g_{i,j}$	Channel gain between SMD $U_{i,j}$ and SeNB π_i
H	Number of heterogeneous cores of a SMD
$I_{i,j}$	Interference at the SMD $U_{i,j}$
$L_{i,j}$	Execution location vector associated with application $G_{i,j}$
$loc_{i,j,k}$	Execution location of task $v_{i,j,k}$
M	Number of the computing frequency levels on a core
N_i^{SMD}	Number of SMDs in the i-th small cell associated with SeNB π_i
N_{total}^{SMD}	Total number of SMDs in the MEC system
N_{total}^{task}	Total number of tasks in the MEC system
$N_{channel}$	Number of channels offered by the MEC system
$N_{i,j}$	Total number of tasks in application $G_{i,j}$
$O_{i,j}$	Execution order vector associated with application $G_{i,j}$
$o_{i,j,k}$	Output data size of task $v_{i,j,k}$
$u_{i,j,k}$	The k-th task to be executed in $O_{i,j}$
$p_{i,j,h}^{actual}$	Actual power consumption of the h-th core of SMD $U_{i,j}$
$p_{i,j,h}^{max}$	Maximum power consumption of the h-th core of SMD $U_{i,j}$
$p_{i,j}^{rxd}$	Power consumption when SMD $U_{i,j}$ receives data
$p_{i,j}^{txd}$	Power consumption when SMD $U_{i,j}$ offloads tasks
$pre(v_{i,j,k})$	Set of the immediate predecessors of task $v_{i,j,k}$
$R_{i,j}$	Achievable uplink transmission rate of SMD $U_{i,j}$
S	Number of SeNBs in the MEC system
$SOL(G_{i,j})$	Computation offloading solution associated with application $G_{i,j}$
$SOL(\pi_i)$	Computation offloading solution associated with SeNB π_i
$suc(v_{i,j,k})$	Set of the immediate successors of task $v_{i,j,k}$
$U_{i,j}$	The j-th SMD in the i-th small cell associated with SeNB π_i
$V_{i,j}$	Set of the tasks in application $G_{i,j}$
$v_{i,j,k}$	The k-th task in application $G_{i,j}$
$\boldsymbol{x}$	A solution to the MCOP
α_t	The t-th frequency scaling factor
π_i	The i-th SeNB in the MEC system
σ^2	Noise power
ω	Bandwidth of a wireless channel

C. Communication Model

When a task is selected for offloading, its associated input data is transmitted to the MEC server via SeNB and MeNB. The transmission delay from SeNB to MEC server via wired connections is usually trivial, thus is ignored. The transmission delay between the corresponding SMD and SeNB is considered in the model.

Let B_{total} and $N_{channel}$ be the total bandwidth and the number of channels offered by the MEC system, respectively. Each channel is of a bandwidth $\omega = B_{total} / N_{channel}$. Each SMD uses one of the $N_{channel}$ channels for data offloading. To guarantee that all SMDs within the same small cell can perform independent computation offloading, it is assumed that $N_{channel}$ is no less than the maximum number of SMDs allowed in a small cell. If two SMDs from neighboring small cells use the same channel to transmit data, interference occurs and the transmission rate is reduced.

According to the Shannon-Harley theorem, it is possible that no errors occur in a channel with limited bandwidth and Gaussian white noise interference if transmitting information at the theoretical maximum transmission rate. In this paper, we set the achievable uplink transmission rate in a channel to the theoretical maximum transmission rate of that channel. Assume each wireless channel is symmetric, i.e. the achievable uplink and downlink transmission rates are the same. When SMD $U_{i,j}$ offloads tasks to the MEC server, the achievable uplink transmission rate $R_{i,j}$ is calculated using Eq. (1).

$$R_{i,j} = \omega \log_2 \left(1 + \frac{p_{i,j}^{txd} g_{i,j}}{\sigma^2 + I_{i,j}} \right) \tag{1}$$

where ω is the channel bandwidth. $p_{i,j}^{txd}$ is the power consumption of $U_{i,j}$ when tasks are offloaded to the MEC server. $g_{i,j}$ is the channel gain between SMD $U_{i,j}$ and SeNB π_i. σ^2 is the noise power. $I_{i,j}$ is the interference parameter associated with SMD $U_{i,j}$ indicating how severe the channel sharing is, as defined in Eq. (2).

$$I_{i,j} = \sum_{l=1, l \neq i}^{S} \sum_{k=1}^{N_l^{SMD}} \lambda_{(i,j),(l,k)} p_{l,k}^{txd} g_{i,(l,k)} \tag{2}$$

where $\lambda_{(i,j),(l,k)} \in \{0,1\}$ is a channel sharing coefficient. $\lambda_{(i,j),(l,k)} = 1$ represents that the same channel is being shared by both $U_{i,j}$ and $U_{l,k}$, and $\lambda_{(i,j),(l,k)} = 0$ otherwise. $p_{l,k}^{txd}$ is the power consumption of $U_{l,k}$ when offloading tasks, and $g_{i,(l,k)}$ is the channel gain between $U_{l,k}$ and π_i, where $U_{l,k}$ is associated with SeNB π_l, $l, i \in \{1,\ldots,S\}$ and $l \neq i$.

D. Local Computing

In this paper, the local computing model is based on the local scheduling model in MCC [25], where the DVFS technique is enabled. For each SMD in the MEC system, assume there are H heterogeneous cores in its processor. This enables the processor execute its tasks in parallel if there are no task-precedence constraints among them. All cores are DVFS enabled, allowing each core to run at different frequency levels at different times. An arbitrary SMD $U_{i,j}$ can be defined as a 4-tuple set $U_{i,j} = (f_{i,j,h}^{max}, p_{i,j,h}^{max}, p_{i,j}^{txd}, p_{i,j}^{rxd})$, where $f_{i,j,h}^{max}$ is the maximum computing frequency on the h-th core of $U_{i,j}$, $h = 1,\ldots,H$, $p_{i,j,h}^{max}$ is the maximum power consumption when the h-th core is working at frequency $f_{i,j,h}^{max}$, $p_{i,j}^{txd}$ is the power consumption of $U_{i,j}$ when offloading tasks, and $p_{i,j}^{rxd}$ is the power consumption of $U_{i,j}$ when receiving data from its associated SeNB.

Assume there are M frequency scaling factors, i.e. $\alpha_1,\ldots,\alpha_M$, for an arbitrary core in an arbitrary SMD, where $0<\alpha_1<\ldots<\alpha_t<\ldots<\alpha_M = 1$ [25]. The actual computing frequency that the h-th core of SMD $U_{i,j}$ is working at can be defined as $f_{i,j,h}^{actual} = \alpha_t \cdot f_{i,j,h}^{max}$, $t = 1,\ldots,M$. The actual power consumption of the h-th core of $U_{i,j}$, $p_{i,j,h}^{actual}$, is equal to $a_{i,j,h} \cdot (f_{i,j,h}^{actual})^{\gamma}$, where γ is a constant in the range of [2, 3] and $a_{i,j,h}$ is a coefficient associated with the chip structure.

Let $T_{SMD,h}^{min}(v_{i,j,k})$ denote the minimum execution time of task $v_{i,j,k}$ if $v_{i,j,k}$ is executed on the h-th core of SMD $U_{i,j}$, at the maximum computing frequency $f_{i,j,h}^{max}$. $T_{SMD,h}^{min}(v_{i,j,k})$ depends on the number of CPU cycles required to perform $v_{i,j,k}$, $c_{i,j,k}$, and the maximum computing frequency, $f_{i,j,h}^{max}$. Then, the actual execution time of $v_{i,j,k}$ on the h-th core $T_{SMD,h}(v_{i,j,k})$ is obtained using Eq. (3) [25].

$$\begin{aligned} T_{SMD,h}(v_{i,j,k}) &= T_{SMD,h}^{min}(v_{i,j,k}) / \alpha_t \\ &= \frac{c_{i,j,k}}{f_{i,j,h}^{max} \cdot \alpha_t} \end{aligned} \tag{3}$$

If task $v_{i,j,k}$ is executed locally on the h-th core of SMD $U_{i,j}$, its actual energy consumption $E_{SMD,h}^{actual}(v_{i,j,k})$ is obtained via Eq. (4), according to [25].

$$\begin{aligned} E_{SMD,h}^{actual}(v_{i,j,k}) &= p_{i,j,h}^{actual} \cdot T_{SMD,h}(v_{i,j,k}) \\ &= a_{i,j,h} \cdot (f_{i,j,h}^{actual})^{\gamma} \cdot (T_{SMD,h}^{min}(v_{i,j,k}) / \alpha_t) \\ &= a_{i,j,h} \cdot (\alpha_t \cdot f_{i,j,h}^{max})^{\gamma} \cdot (T_{SMD,h}^{min}(v_{i,j,k}) / \alpha_t) \\ &= (\alpha_t)^{\gamma-1} \cdot a_{i,j,h} \cdot (f_{i,j,h}^{max})^{\gamma} \cdot T_{SMD,h}^{min}(v_{i,j,k}) \\ &= (\alpha_t)^{\gamma-1} \cdot E_{SMD,h}^{max}(v_{i,j,k}) \\ &= (\alpha_t)^{\gamma-1} \cdot p_{i,j,h}^{max} \cdot T_{SMD,h}^{min}(v_{i,j,k}) \end{aligned} \tag{4}$$

where $E_{SMD,h}^{max}(v_{i,j,k})$ is the maximum energy consumption if task $v_{i,j,k}$ is executed on the h-th core of SMD $U_{i,j}$ at the maximum computing frequency.

Before executing task $v_{i,j,k}$, execution of all its immediate predecessors must be completed. If $v_{i,j,k}$ is to be launched on a core of SMD $U_{i,j}$, the ready time for executing it is related to the completion times of its immediate predecessors in $pre(v_{i,j,k})$. That is, the ready time for executing $v_{i,j,k}$, $RT_{SMD}^{exe}(v_{i,j,k})$, depends on the maximum completion time of all tasks in $pre(v_{i,j,k})$. Assume $v_{i,j,l} \in pre(v_{i,j,k})$ is an immediate predecessor of $v_{i,j,k}$ which can be executed on either the local SMD or the MEC server. Let $CT_{SMD}^{exe}(v_{i,j,l})$ be the completion time of task $v_{i,j,l}$ if it is executed on SMD $U_{i,j}$. Let $CT_{SMD}^{rxd}(v_{i,j,l})$ be the completion time that $U_{i,j}$ finishes receiving the output data of $v_{i,j,l}$ from wireless channel if $v_{i,j,l}$ is executed on the MEC server. The definition of $RT_{SMD}^{exe}(v_{i,j,k})$ is expressed in Eq. (5).

$$RT_{SMD}^{exe}(v_{i,j,k}) = \max_{v_{i,j,l}\in pre(v_{i,j,k})} \max\left\{CT_{SMD}^{exe}(v_{i,j,l}), CT_{SMD}^{rxd}(v_{i,j,l})\right\} \quad (5)$$

where $CT_{SMD}^{exe}(v_{i,j,l})=0$ means $v_{i,j,l}$ is offloaded to the MEC server and $CT_{SMD}^{rxd}(v_{i,j,l})=0$ means $v_{i,j,l}$ is executed locally. So, in any case, between $CT_{SMD}^{exe}(v_{i,j,l})$ and $CT_{SMD}^{rxd}(v_{i,j,l})$, one of them is set to 0.

If task $v_{i,j,k}$ is selected to run on a core of $U_{i,j}$, its execution cannot start before its ready time $RT_{SMD}^{exe}(v_{i,j,k})$. It is possible that $v_{i,j,k}$ is executed sometime after $RT_{SMD}^{exe}(v_{i,j,k})$, because that core may be busy with executing other tasks at that time. Let the start time for executing $v_{i,j,k}$ denoted by $ST_{SMD}^{exe}(v_{i,j,k})$, $ST_{SMD}^{exe}(v_{i,j,k}) \geq RT_{SMD}^{exe}(v_{i,j,k})$ [25].

E. Edge Computing

The edge computing model in this paper is based on the cloud scheduling model in [25]. However, the authors assume there is only one active SMD in the MCC network, which is not realistic. We assume there are multiple SMDs in the MEC system, which mimics the demands from the real world.

To model the offloading task $v_{i,j,k}$ to the MEC server, let $RT_{SMD}^{txd}(v_{i,j,k})$ be the ready time for transmitting $v_{i,j,k}$ from $U_{i,j}$ via wireless channel. If $v_{i,j,k}$ is to be offloaded to the MEC server, the ready time for transmitting it, $RT_{SMD}^{txd}(v_{i,j,k})$, depends on the maximum completion time of all tasks in $pre(v_{i,j,k})$. Let $CT_{SMD}^{txd}(v_{i,j,l})$ be the completion time that $U_{i,j}$ finishes transmitting $v_{i,j,l} \in pre(v_{i,j,k})$ to the MEC server. The expression of $RT_{SMD}^{txd}(v_{i,j,k})$ is presented in Eq. (6) [25].

$$RT_{SMD}^{txd}(v_{i,j,k}) = \max_{v_{i,j,l}\in pre(v_{i,j,k})} \max\left\{CT_{SMD}^{exe}(v_{i,j,l}), CT_{SMD}^{txd}(v_{i,j,l})\right\} \quad (6)$$

where $CT_{SMD}^{exe}(v_{i,j,l})=0$ if $v_{i,j,l}$ is offloaded to the MEC server and $CT_{SMD}^{txd}(v_{i,j,l})=0$ if $v_{i,j,l}$ is executed locally.

Let the time duration required to transmit $v_{i,j,k}$ to the MEC server be $T_{SMD}^{txd}(v_{i,j,k})$, as defined in Eq. (7).

$$T_{SMD}^{txd}(v_{i,j,k}) = \frac{d_{i,j,k}}{R_{i,j}} \quad (7)$$

where $d_{i,j,k}$ and $R_{i,j}$ are the input data size of $v_{i,j,k}$ and the achievable uplink transmission rate, respectively.

If task $v_{i,j,k}$ is offloaded to the MEC server, the energy consumption of $U_{i,j}$ for transmitting this task, $E_{SMD}^{txd}(v_{i,j,k})$, is expressed in Eq. (8).

$$E_{SMD}^{txd}(v_{i,j,k}) = p_{i,j}^{txd} \cdot T_{SMD}^{txd}(v_{i,j,k}) \quad (8)$$

Let $CT_{MEC}^{exe}(v_{i,j,l})$ be the completion time of $v_{i,j,l} \in pre(v_{i,j,k})$ if $v_{i,j,l}$ is executed on the MEC server. The ready time for executing $v_{i,j,k}$ on the MEC server, $RT_{MEC}^{exe}(v_{i,j,k})$, is defined in Eq. (9).

$$RT_{MEC}^{exe}(v_{i,j,k}) = \max\{CT_{SMD}^{txd}(v_{i,j,k}), \max_{v_{i,j,l}\in pre(v_{i,j,k})} CT_{MEC}^{exe}(v_{i,j,l})\} \quad (9)$$

Let $T_{MEC}^{exe}(v_{i,j,k})$ denote the execution time of $v_{i,j,k}$ on the MEC server, as defined in Eq. (10).

$$T_{MEC}^{exe}(v_{i,j,k}) = \frac{c_{i,j,k}}{F} \quad (10)$$

where $c_{i,j,k}$ is the number of CPU cycles required to execute $v_{i,j,k}$, and F is the computing capability of the MEC server.

The MEC server can start to transmit the output data of $v_{i,j,k}$ back to $U_{i,j}$, immediately after the completion of $v_{i,j,k}$. Let $RT_{MEC}^{txd}(v_{i,j,k})$ be the ready time for the MEC server to transmit back the output data of $v_{i,j,k}$, as defined in Eq. (11).

$$RT_{MEC}^{txd}(v_{i,j,k}) = CT_{MEC}^{exe}(v_{i,j,k}) \quad (11)$$

Let the time duration required to receive the output data of $v_{i,j,k}$ from the MEC server, denoted by $T_{SMD}^{rxd}(v_{i,j,k})$, as defined in Eq. (12).

$$T_{SMD}^{rxd}(v_{i,j,k}) = \frac{o_{i,j,k}}{R_{i,j}} \quad (12)$$

where $o_{i,j,k}$ is the output data size after the execution of $v_{i,j,k}$.

In [25], the cloud scheduling model does not consider the energy consumption of SMD $U_{i,j}$ incurred when receiving the output data of task $v_{i,j,k}$, which is not practical. In contrast, our edge computing model takes the energy consumption of receiving the output data from MEC server into consideration, which helps to accurately estimate the energy consumption. The energy consumption of $U_{i,j}$ for receiving the output data of $v_{i,j,k}$, $E_{SMD}^{rxd}(v_{i,j,k})$, is defined in Eq. (13), according to [22][38].

$$E_{SMD}^{rxd}(v_{i,j,k}) = p_{i,j}^{rxd} \cdot T_{SMD}^{rxd}(v_{i,j,k}) \quad (13)$$

Taking the task graph in Fig. 2 as an example, we briefly explain the process of local and edge computing. In the MEC system, there is one SeNB, namely π_1, and one SMD associated with π_1, namely $U_{1,1}$. The application $G_{1,1}$ has seven tasks, i.e. $v_{1,1,k}$, k = 1,…,7, to be run on $U_{1,1}$. Suppose $U_{1,1}$ owns three cores (i.e. h = 1,2,3). Table II shows the execution time of each task on different cores and the actual execution locations of all tasks. Let $loc_{1,1,k} \in \{1,2,3,4\}$ be the execution location of $v_{1,1,k}$, k = 1,…,7. If $1 \leq loc_{1,1,k} \leq 3$, $v_{1,1,k}$ is executed on the $loc_{1,1,k}$-th core of $U_{1,1}$. If $loc_{1,1,k} = 4$, $v_{1,1,k}$ is offloaded to the MEC server. For simplicity, we set the time duration required to transmit each task to the MEC server, $T_{SMD}^{txd}(v_{1,1,k}) = 3$, the time duration required to receive the output data of each task from the MEC server, $T^{rxd}_{SMD}(v_{1,1,k})=1$, and the execution time of each task on the MEC server, $T^{exe}_{MEC}(v_{1,1,k})=1$, $k=1,\ldots,7$.

TABLE II
EXECUTION TIMES AND LOCATIONS OF ALL TASKS

Task	$T^{min}_{SMD,1}$	$T^{min}_{SMD,2}$	$T^{min}_{SMD,3}$	Location
v_1	1	3	4	3
v_2	3	4	5	1
v_3	1	2	4	4
v_4	3	5	7	1
v_5	2	5	6	2
v_6	2	4	6	3
v_7	1	3	4	4

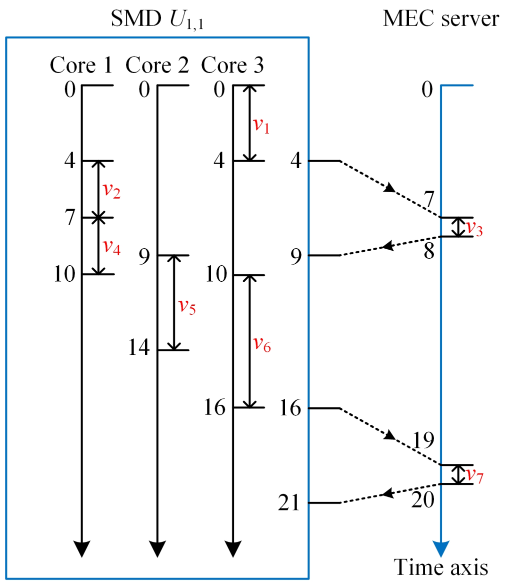


Fig. 3. Task scheduling plan.

Fig. 3 shows the task scheduling plan according to Table II. The start time and completion time of task v_6 are $ST^{exe}_{SMD}(v_6)=10$ and $CT^{exe}_{SMD}(v_6)=16$, respectively. The completion time of $G_{1,1}$, $CT(G_{1,1})=CT^{rxd}_{SMD}(v_7)$, is 21.

F. Problem Formulation

The new MCOP model aims to simultaneously minimize the average completion time of applications and the average energy consumption of SMDs in the above MEC system.

Let $CT^{exe}_{SMD}(v_{i,j,end})$ and $CT^{rxd}_{SMD}(v_{i,j,end})$ be the completion time for executing the end task $v_{i,j,end}$ in application $G_{i,j}$ on SMD $U_{i,j}$ and that for receiving the output data of $v_{i,j,end}$ via wireless channel, respectively. Let $\varepsilon_{i,j,end}$ be a binary variable indicating if $v_{i,j,end}$ is executed on $U_{i,j}$ or the MEC server. $\varepsilon_{i,j,end}=1$ means $v_{i,j,end}$ is executed on $U_{i,j}$, and $\varepsilon_{i,j,end}=0$ otherwise. The completion time of application $G_{i,j}$ on SMD $U_{i,j}$, $CT(G_{i,j})$, is defined in Eq. (14), which is equal to the completion time of the end task $v_{i,j,end}$ in $G_{i,j}$. If $v_{i,j,end}$ is offloaded to the MEC server, $CT(G_{i,j})$ equals to the time when the output data of $v_{i,j,end}$ is fully received. Otherwise, $CT(G_{i,j})$ equals to the time when the execution of $v_{i,j,end}$ is over.

$$CT(G_{i,j})= \\ \varepsilon_{i,j,end}\cdot CT^{exe}_{SMD}(v_{i,j,end})+(1-\varepsilon_{i,j,end})\cdot CT^{rxd}_{SMD}(v_{i,j,end}) \tag{14}$$

With the obtained completion times of all applications on all SMDs, the average completion time (ACT) of applications in the MEC system can be calculated using Eq. (15).

$$ACT=\frac{1}{N^{SMD}_{total}}\sum_{i=1}^{S}\sum_{j=1}^{N_i^{SMD}}CT(G_{i,j}) \tag{15}$$

where $N^{SMD}_{total}=\sum_{i=1}^{S}N_i^{SMD}$ is the total number of SMDs in the MEC system.

The average energy consumption (AEC) of all SMDs in the MEC system can be obtained by Eq. (16).

$$AEC=\frac{1}{N^{task}_{total}}\sum_{i=1}^{S}\sum_{j=1}^{N_i^{SMD}}\sum_{k=1}^{N_{i,j}}E(v_{i,j,k}) \tag{16}$$

where

$$E(v_{i,j,k})=\varepsilon_{i,j,k}\cdot E^{actual}_{SMD,h}(v_{i,j,k})+ \\ (1-\varepsilon_{i,j,k})\cdot(E^{txd}_{SMD}(v_{i,j,k})+E^{rxd}_{SMD}(v_{i,j,k})) \tag{17}$$

$N^{task}_{total}=\sum_{i=1}^{S}\sum_{j=1}^{N_i^{SMD}}N_{i,j}$ is the total number of tasks in the MEC system. $\varepsilon_{i,j,k}=1$ if $v_{i,j,k}$ is executed on SMD $U_{i,j}$; $\varepsilon_{i,j,k}=0$, otherwise.

The MCOP can be defined as a bi-objective MOP problem, minimizing ACT in Eq. (15) and AEC in Eq. (16) subject to all task precedence constraints as defined in Eq. (18).

$$\underset{L_{i,j},O_{i,j}}{\text{Minimize}}\ (ACT,\ AEC)$$

s.t.

$$\begin{aligned}
&C1:\ order(v_{i,j,k})<order(v_{i,j,l}),\ \text{if}\ e(v_{i,j,k},v_{i,j,l})\in E_{i,j}\\
&C2:\ CT^{exe}_{SMD}(v_{i,j,l})\le RT^{exe}_{SMD}(v_{i,j,k}),\ \forall v_{i,j,l}\in pre(v_{i,j,k})\\
&C3:\ CT^{rxd}_{SMD}(v_{i,j,l})\le RT^{exe}_{SMD}(v_{i,j,k}),\ \forall v_{i,j,l}\in pre(v_{i,j,k})\\
&C4:\ CT^{txd}_{SMD}(v_{i,j,k})\le RT^{exe}_{MEC}(v_{i,j,k})\\
&C5:\ \max_{v_{i,j,l}\in pre(v_{i,j,k})}CT^{exe}_{MEC}(v_{i,j,l})\le RT^{exe}_{MEC}(v_{i,j,k})\\
&C6:\ loc_{i,j,k}\in\{0,1,\ldots,H\},\ \forall i,j,k
\end{aligned} \tag{18}$$

where, constraint C1 is the execution order constraint between two tasks, i.e. if $e(v_{i,j,k},v_{i,j,l})\in E_{i,j}$, task $v_{i,j,l}$ cannot be executed before the completion of task $v_{i,j,k}$. Constraints C2 and C3 are the local task-precedence constraints, ensuring that $v_{i,j,k}$ cannot be executed before all its immediate predecessors are completed. Constraints C4 and C5 are the edge task-precedence constraints, indicating that $v_{i,j,k}$ cannot be executed before it is completely offloaded to the MEC server and all its immediate predecessors are completed on the MEC server. Constraint C6 is a computation offloading execution location constraint, specifying where $v_{i,j,k}$ is executed, i.e. which core of $U_{i,j}$ or the MEC server.

IV. Overview of MOP and MOEA/D

A. MOP

A MOP can be defined as Eq. (19).

$$\begin{cases}\text{Minimize} \quad F(\boldsymbol{x}) = (f_1(\boldsymbol{x}),...,f_m(\boldsymbol{x}))^T \\ \text{Subject to} \quad \boldsymbol{x} \in \Omega\end{cases} \tag{19}$$

where $\boldsymbol{x} = (x_1,...,x_n)$ is an n-dimension decision vector in search space Ω. There are m objective functions in objective vector $F(\boldsymbol{x})$, where $f_1(\boldsymbol{x}),\ldots,f_m(\boldsymbol{x})$ conflict with each other [39].

Given two different solutions $\boldsymbol{x}_1$, $\boldsymbol{x}_2 \in \Omega$, $\boldsymbol{x}_1$ is said to dominate $\boldsymbol{x}_2$, if $f_i(\boldsymbol{x}_1) \le f_i(\boldsymbol{x}_2)$ for all $i \in \{1,\ldots,m\}$, and $f_j(\boldsymbol{x}_1) < f_j(\boldsymbol{x}_2)$ for at least one $j \in \{1,\ldots,m\}$. A solution $\boldsymbol{x}^* \in \Omega$ is known as Pareto optimal if no other solution in Ω dominates it. The set of all Pareto optimal solutions is known as the Pareto optimal set, of which the mapping in the objective space is known as the Pareto optimal front (PF).

There are mainly two methods to handle a MOP. One is to convert it to a single-objective optimization problem (SOP) by objective aggregation. In this case, the commonly used method is weighted sum, where each objective, e.g. the ACT and AEC in this paper, is assigned a weight. However, weight values should be set in advance. Heuristics and metaheuristics (including EAs) are often used to address a SOP. By running them once, a single solution is output. If system demands change, the weight values need to be re-set. Hence, the first method only obtains a compromised solution, which cannot reflect the conflicting features between objectives. The other method to tackle MOP is to use MOEAs. Any MOEA is capable of obtaining a set of nondominated solutions in a single run. These solutions reflect the Pareto-dominance relation among them. This is what a decision maker expects to know. Even if the system demands change, the nondominated solutions obtained by MOEAs are still valid. This is why MOEAs are more appropriate to address the MCOP problem.

Currently, MOEAs have attracted increasingly more research attention and been successfully applied to address various MOP problems, such as computation offloading [27][30], workflow scheduling [20], function optimization [7][40], feature selection [41], and job shop scheduling [42].

Pareto-dominance based MOEAs are the mainstream optimizers in the literature, such as, NSGA-II. Nevertheless, they usually suffer from prematurity and local optima. On the other hand, compared with them, MOEA/D has been reported to achieve better global exploration ability with lower computational overhead [31][40]. This motivates us to adapt MOEA/D for the new MCOP.

B. Original MOEA/D

MOEA/D has been applied to various MOPs due to its high effectiveness yet low computational cost [31][32][33][40][43][44]. MOEA/D decomposes a MOP into a number of scalar optimization subproblems (SOSPs) that are simultaneously optimized in a collaborative and time efficient manner. It employs genetic operators to generate new solutions and obtain a set of nondominated solutions through an evolution process. Three basic methods have been employed in the literature for decomposition, among which the Tchebycheff method is the mostly used, and adopted in our proposed algorithm.

Let $\boldsymbol{\lambda}^1,...,\boldsymbol{\lambda}^{N_P}$ be a set of uniformly spread weighted vectors, where N_P is the number of SOSPs. $\boldsymbol{\lambda}^i = (\lambda_1^i,...,\lambda_m^i)$ is the weight vector associated with the i-th SOSP, $i = 1,\ldots,N_P$, where m is the number of objectives, and $\sum_{j=1}^m \lambda_j^i = 1$. Let $\boldsymbol{z}^* = (z_1^*,...,z_m^*)$ be the reference point, where $z_j^* = \min\{f_j(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\}$, and $j = 1,\ldots,m$. Based on the Tchebycheff method, the i-th SOSP is defined by Eq. (20).

$$\begin{cases}\text{Minimize} \quad g^{te}(\boldsymbol{x} \mid \boldsymbol{\lambda}^i, \boldsymbol{z}^*) = \max\limits_{1\le j\le m}\{\lambda_j^i \mid f_j(\boldsymbol{x}) - z_j^* |\} \\ \text{Subject to} \quad \boldsymbol{x} \in \Omega\end{cases} \tag{20}$$

V. The Proposed MOEA/D for MCOP

This paper proposes an improved MOEA/D to address the new MCOP, namely MOEA/D-MCOP. This section first introduces the solution representation and evaluation. Two specially devised performance enhancing schemes, i.e. a problem-specific population initialization scheme, and a DVFS-based energy conservation scheme are then explained. Then, the overall procedure of MOEA/D-MCOP is given in detail. The complexity of MOEA/D-MCOP is analyzed at last.

A. Solution Representation and Evaluation

As mentioned in Section III, for application $G_{i,j}$ on SMD $U_{i,j}$, the execution locations and the execution orders of all tasks in $G_{i,j}$ need to be determined in the MCOP. Let $V_{i,j} = \{v_{i,j,1},...,v_{i,j,N_{i,j}}\}$ denote the set of tasks in $G_{i,j}$. Let $L_{i,j} = (loc_{i,j,1},...,loc_{i,j,k},...,loc_{i,j,N_{i,j}})$ denote the execution location vector of all tasks in $G_{i,j}$, where $loc_{i,j,k} \in \{1,...,H,H+1\}$ is the execution location of task $v_{i,j,k}$, and H is the number of cores in $U_{i,j}$. If $1 \le loc_{i,j,k} \le H$, $v_{i,j,k}$ is executed on the $loc_{i,j,k}$-th core of $U_{i,j}$. If $loc_{i,j,k} = H+1$, $v_{i,j,k}$ is offloaded to the MEC server. Let $O_{i,j} = (u_{i,j,1},...,u_{i,j,k},...,u_{i,j,N_{i,j}})$ denote the execution order vector of all tasks in $G_{i,j}$, where $u_{i,j,k} \in V_{i,j}$ denotes the k-th task to be executed in $O_{i,j}$, e.g. $u_{i,j,1} = v_{i,j,start}$ and $u_{i,j,N_{i,j}} = v_{i,j,end}$.

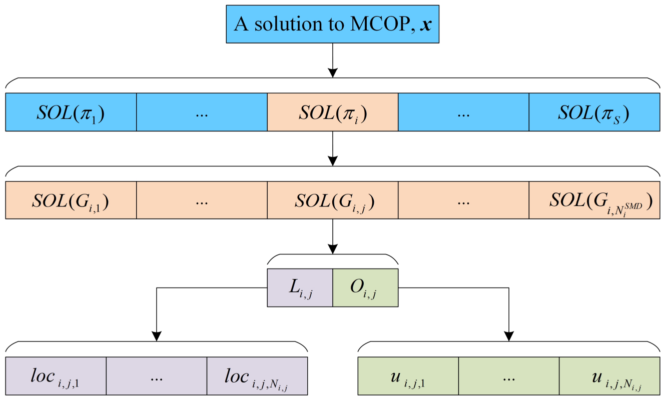


Fig. 4. Solution representation in the MCOP.

Let $SOL(G_{i,j}) = (L_{i,j}, O_{i,j})$ be a computation offloading solution (SOL) for application $G_{i,j}$, $i = 1,\ldots,S$, $j = 1,\ldots, N_i^{SMD}$. Let $SOL(\pi_i) = (SOL(G_{i,1}),\ldots,SOL(G_{i,N_i^{SMD}}))$ be a SOL for all applications on all SMDs associated with SeNB π_i, $i = 1,\ldots,S$. A solution to the MCOP, $\boldsymbol{x} = (SOL(\pi_1),\ldots,SOL(\pi_S))$, consists of all SOLs associated with all SeNBs in the MEC system. Fig. 4 shows an example solution to the MCOP.

Given a solution $\boldsymbol{x}$, its objective function values, $F(\boldsymbol{x}) = (f_{ACT}(\boldsymbol{x}), f_{AEC}(\boldsymbol{x}))$, can be calculated using Eqs. (15) and (16) in Section III-F.

B. Problem-Specific Population Intialization Scheme

The problem-specific population initialization scheme is based on two methods, including a latency-based execution location initialization method and a commonly used execution order initialization method.

1) Latency-Based Execution Location Initialization

Initial population usually has a significant impact on the optimization performance of a MOEA. An effective population initialization scheme helps to guide a MOEA towards promising areas in the search space [40]. Randomly generated initial population may have better diversity, they are, however, not always helpful for the search to quickly locate areas with high-quality solutions in exponential search spaces. In particular, for highly constrained optimization problems, misleading search directions of a MOEA might lead to serious deterioration on the optimization performance [31].

To the best of our knowledge, most of EAs for COP problems initialize execution locations for all tasks in a random manner [20][27]. For most of small scale COP problems, random location generation helps to diversify the population and has a positive influence on the optimization performance. However, this method is no longer applicable to the highly constrained large scale MCOP problem concerned in this paper, due to the large number of tasks involved and the task precedence constraints.

The proposed latency-based execution location initialization (LELI) method decides if a task is executed locally or offloaded to the MEC server by comparing its average computing time if it is executed on SMD, and its task offloading time if it is executed on the MEC server. As aforementioned, a solution to the MCOP problem, $\boldsymbol{x}$, contains all execution locations of all tasks in the MEC system and the execution orders among them. By reducing the completion time of each task in a greedy manner, this method reduces the completion time of each application, which also helps to reduce the average completion time of all applications in the MEC system, i.e. ACT.

Let $T_{SMD}^{avg}(v_{i,j,k})$ and $T_{MEC}^{ofld}(v_{i,j,k})$ be the average execution time and the task offloading time of $v_{i,j,k}$, respectively. For an arbitrary application $G_{i,j}$, the procedure to determine the execution locations of all tasks is described as below.

For each $v_{i,j,k}$ in $G_{i,j}$, $T_{SMD}^{avg}(v_{i,j,k})$ and $T_{MEC}^{ofld}(v_{i,j,k})$ are calculated. $T_{MEC}^{ofld}(v_{i,j,k})$ is the summation of $T_{SMD}^{txd}(v_{i,j,k})$, $T_{MEC}^{exe}(v_{i,j,k})$ and $T_{SMD}^{rxd}(v_{i,j,k})$ (see Eqs. (7), (10) and (12)). If $T_{SMD}^{avg}(v_{i,j,k}) < T_{MEC}^{ofld}(v_{i,j,k})$, $v_{i,j,k}$ is executed on a randomly selected core of $U_{i,j}$; otherwise, $v_{i,j,k}$ is offloaded to the MEC server. The execution location initialization for all tasks in $G_{i,j}$ is shown in Algorithm 1, where *randInt*(1, H) is an integer randomly generated in the range [1, H].

Algorithm 1. Latency-Based Execution Location Initialization

```
Input: application $G_{i,j}$.
Output: execution location vector $L_{i,j}$.
1.  Initialize vector $L_{i,j} = (loc_{i,j,1},\ldots,loc_{i,j,N_{i,j}})$;
2.  for $k = 1$ to $N_{i,j}$ do
3.      Calculate $T_{SMD}^{avg}(v_{i,j,k}) = (1/H)\cdot\sum_{h=1}^{H} c_{i,j,k} / f_{i,j,h}^{max}$;
4.      Calculate $T_{MEC}^{ofld}(v_{i,j,k}) = T_{SMD}^{txd}(v_{i,j,k}) + T_{MEC}^{exe}(v_{i,j,k}) + T_{SMD}^{rxd}(v_{i,j,k})$;
5.      if $T_{SMD}^{avg}(v_{i,j,k}) < T_{MEC}^{ofld}(v_{i,j,k})$ then
6.          Generate a random integer randInt(1, H);
7.          Set $loc_{i,j,k}$ = randInt(1, H);        // local computing
8.      else
9.          Set $loc_{i,j,k} = H + 1$;                  // edge computing
```

2) Random-Selection-Based Execution Order Initialization

In [20], an efficient execution order initialization method based on random selection is proposed, i.e. random-selection-based execution order initialization (RSEOI), where a task set, Ψ, maintains all those tasks which are not sorted but their immediate predecessors are sorted. A task $v_{i,j,r}$ is randomly selected from Ψ and added to the end of execution order vector $O_{i,j}$. After that, $v_{i,j,r}$ is removed from Ψ, and its immediate successors, whose immediate predecessors are all sorted, are inserted into Ψ. Once all tasks in $G_{i,j}$ are sorted, the task selection process stops and $O_{i,j}$ is returned as the output.

By running the RSEOI method multiple times, a set of different execution order vectors for $G_{i,j}$ can be obtained. RSEOI is thus incorporated into the PSPI scheme to diversify the initial population. The execution order initialization for all tasks in $G_{i,j}$ is shown in Algorithm 2.

Algorithm 2. Random-Selection-Based Execution Order Initialization

```
Input: application $G_{i,j}$.
Output: execution order vector $O_{i,j}$ .
1.  Initialize vector $O_{i,j} = (u_{i,j,1},...,u_{i,j,N_{i,j}})$ ;
2.  Set the sortable task set $\Psi = \emptyset$;
3.  Set the sorted task set $Z = \emptyset$;
4.  Set $\Gamma = V_{i,j} - \{v_{i,j,start}\}$;
5.  Set $u_{i,j,1} = v_{i,j,start}$ and $Z = Z \cup \{v_{i,j,start}\}$;
6.  Set $index = 1$;                    // index of the current task in $O_{i,j}$
7.  while $\Gamma \neq \emptyset$ do
8.      for $v_{i,j,k} \in \Gamma$ do
9.          if $pre(v_{i,j,k}) \subset Z$ and $v_{i,j,k} \notin \Psi$ then
10.             Set $\Psi = \Psi \cup \{v_{i,j,k}\}$;
11.     Randomly select a task $v_{i,j,r}$ from $\Psi$;
12.     Set $\Psi = \Psi - \{v_{i,j,r}\}$, $Z = Z \cup \{v_{i,j,r}\}$ and $\Gamma = \Gamma - \{v_{i,j,r}\}$;
13.     Set $index = index + 1$ and $u_{i,j,index} = v_{i,j,r}$;
```

3) Overall Procedure of the Problem-Specific Population Initialization Scheme

The problem-specific population initialization (PSPI) scheme is based on LELI and RSEOI. The pseudocode of PSPI is shown in Fig. 5, where *randInt*(1, *H*+1) is an integer randomly generated in the range [1, *H*+1].

The execution locations of all tasks are initialized by random execution location generation in half of the initial population, and LELI (i.e. Algorithm 1) for the other half of the initial population. The random execution location generation introduces certain level of population diversity while LELI provides high-quality solutions for the evolution. The execution order vector associated with each application $G_{i,j}$ is initialized by Algorithm 2.

```
1.  Set $POP = \emptyset$ and $index = 1$;              // $POP$: population
2.  while $index \leq N_P$ do                          // $N_P$: population size
3.      Initialize solution $\boldsymbol{x}_{index} = (SOL(\pi_1),\ldots,SOL(\pi_S))$;
4.      for $i = 1$ to $S$ do                          // for each $SOL(\pi_i)$
5.          for $j = 1$ to $N_i^{SMD}$ do              // for each $SOL(G_{i,j})$
6.              if $index \leq N_P/2$ then
7.                  for $k = 1$ to $N_{i,j}$ do
8.                      Generate a random integer $randInt(1, H+1)$;
9.                      Set $loc_{i,j,k} = randInt(1, H+1)$;
10.             else Obtain $L_{i,j}$ by Algorithm 1;
11.             Obtain $O_{i,j}$ by Algorithm 2;
12.             Set $SOL(G_{i,j}) = (L_{i,j}, O_{i,j})$;
13.     Set $POP = POP \cup \boldsymbol{x}_{index}$ and $index = index + 1$;
14. Output population $POP$.
```

Fig. 5. Pseudocode of the PSPI scheme.

C. DVFS-Based Energy Conservation Scheme

For a given SMD, if a high-performance core and a low-performance core achieve similar computing performance when executing a given task, then executing it on the latter can reduce the energy consumption. The DVFS technique can be utilized to reduce the computing frequency of high-performance cores of SMDs, for energy conservation purpose.

Recently, DVFS has been widely used as a promising power management solution to reduce energy consumption of SMDs in mobile cloud computing [25][27][45][46][47][48]. However, to the best of our knowledge, there is a lack of research applying DVFS to COP and MCOP problems in mobile edge computing. As mentioned in Section III-D, there are *H* heterogeneous cores in each SMD, where each core can run at *M* different computing frequency levels. This paper introduces a DVFS-based energy conservation (DVFS-EC) scheme in the proposed algorithm to further decrease the energy consumption of SMDs.

In [25], a DVFS algorithm is presented for a COP in mobile cloud computing. By dynamically tuning the computing frequency level of each core, this algorithm can significantly reduce the energy consumption of the associated mobile device.

Algorithm 3. DVFS Based on $SOL(G_{i,j})$

```
Input: task scheduling plan associated with $SOL(G_{i,j})$.
Output: new task scheduling plan with new computing
        frequency level assignment for local tasks.
1.  for $k = 1$ to $N_{i,j}$ do
2.      if $1 \leq loc_{i,j,k} \leq H$ then                  // local tasks
3.          Set $flag = 0$ and $t = 1$;
4.          while $flag == 0$ and $t < M$ do
5.              Calculate a new completion time $CT_{SMD}^{exe,new}(v_{i,j,k})$ if
                $v_{i,j,k}$ is executed using the $t$-th computing frequency level;
6.              if there exists next task $v_{i,j,next}$ on the same core then
7.                  Set $limit_1 = ST_{SMD}^{exe}(v_{i,j,next})$ ;
8.              else if $v_{i,j,k}$ is the last task on this core then
9.                  Set $limit_1 = CT(G_{i,j})$;
10.             if $v_{i,j,k}$ is not end task then
11.                 Set $limit_2 = \min_{v_{i,j,l} \in suc(v_{i,j,k})} ST_{SMD}^{exe}(v_{i,j,l})$ ;
12.             else Set $limit_2 = CT(G_{i,j})$;
13.             if $CT_{SMD}^{exe,new}(v_{i,j,k}) \leq limit_1$ and $CT_{SMD}^{exe,new}(v_{i,j,k}) \leq limit_2$ then
14.                 Assign the $t$-th computing frequency level to $v_{i,j,k}$;
15.                 Set $flag = 1$;
16.             Set $t = t+1$;
```

The DVFS algorithm in [25] is adapted for the MCOP formulated in this paper, as shown in Algorithm 3. Given application $G_{i,j}$ with its computation offloading solution $SOL(G_{i,j})$, the associated computation offloading is calculated, including the start time and the completion time of $v_{i,j,k}$, $ST_{SMD}^{exe}(v_{i,j,k})$ and $CT_{SMD}^{exe}(v_{i,j,k})$, and the completion time of $G_{i,j}$, $CT(G_{i,j})$, according to Sections III-D, III-E, and III-F. Algorithm 3 reduces the energy consumption of SMD $U_{i,j}$ by iteratively tuning the computing frequency levels of local cores that are used to execute task(s). The resulting task scheduling plan with a new computing frequency level assignment consumes less energy. Different from the DVFS algorithm in [25] that might lead to a higher completion time, Algorithm 3 does not require additional time for completing $G_{i,j}$.

The pseudocode of the DVFS-EC scheme is shown in Fig. 6. For each application $G_{i,j}$ with a certain $SOL(G_{i,j})$, Algorithm 3 obtains a new computing frequency level assignment. The DVFS-EC scheme aims at reducing the average energy consumption of all SMDs in the MEC system, i.e. AEC, which helps to improve quality of solutions to the MCOP.

```
1.  Given a solution $x$, calculate its associated ACT and AEC values,
    $f_{ACT}(x)$ and $f_{AEC}(x)$;                    // see Eqs. (15) and (16)
2.  for $i = 1$ to $S$ do
3.      for $j = 1$ to $N_i^{SMD}$ do
4.          Run Algorithm 3 based on the task scheduling plan
            associated with $SOL(G_{i,j})$;
5.          for $k = 1$ to $N_{i,j}$ do
6.              if $1 \le loc_{i,j,k} \le H$ then        // local tasks
7.                  Update energy consumption $E_{SMD,h}^{actual}(v_{i,j,k})$;
8.  Update $f_{AEC}(x)$.
```

Fig. 6. Pseudocode of the DVFS-EC scheme.

D. Overall Procedure of MOEA/D-MCOP

The proposed MOEA/D is based on the original MOEA/D (see Section IV-B). Denote the number of neighbors of each SOSP by W. The set of neighbors of the i-th SOSP, $\varphi(i)$, contains W closest neighbors, where the closeness between any two SOSPs is measured by the Euclidean distance between the two corresponding weight vectors. Let **EP** represent the external population storing the nondominated solutions obtained during the evolution.

The overall procedure of the proposed MOEA/D is presented as below.

MOEA/D-MCOP Procedure:

Input:

- N_P: the number of subproblems;
- $\boldsymbol{\lambda}^1,...,\boldsymbol{\lambda}^{N_P}$: uniformly spread weight vectors;
- W: the number of neighbors for each subproblem;
- Stopping condition.

Output:

- **EP**: an external population storing the nondominated solutions obtained during the evolution.

Step 1) Initialization:

Step 1.1) Set **EP** = ∅.

Step 1.2) For each $\boldsymbol{\lambda}^i$, i = 1,…,N_P, obtain W closest weight vectors based on Euclidean distance, $\boldsymbol{\lambda}^{i_1},...,\boldsymbol{\lambda}^{i_W}$, and set $\varphi(i) = \{i_1,...,i_W\}$.

Step 1.3) Generate an initial population, $\boldsymbol{x}_1,...,\boldsymbol{x}_{N_P}$, by using the PSPI scheme in Section V-B and evaluate the objective functions for each solution.

Step 1.4) Initialize reference point $z^* = (z_1^*,...,z_m^*)$.

Step 2) Repeat:

for i = 1 **to** N_P **do**

Step 2.1) Reproduction: Apply crossover (see Algorithm 5) and mutation (see Algorithm 7) operators to generate a new solution $\boldsymbol{y}$ based on $\boldsymbol{x}_k$ and $\boldsymbol{x}_l$, where k, $l \in \varphi(i)$ and $k \neq l$.

Step 2.2) DVFS-EC: Apply the DVFS-EC scheme (see Section V-C) to reduce the AEC value of $\boldsymbol{y}$, $f_{AEC}(\boldsymbol{y})$.

Step 2.3) Update of z^*: If $f_j(\boldsymbol{y}) < z_j^*$, then set $z_j^* = f_j(\boldsymbol{y})$, j = 1,…,m.

Step 2.4) Update of neighboring solutions: For each $q \in \varphi(i)$, if $g^{te}(\boldsymbol{y} \mid \boldsymbol{\lambda}^q, z^*) \le g^{te}(\boldsymbol{x}_q \mid \boldsymbol{\lambda}^q, z^*)$, then set $\boldsymbol{x}_q = \boldsymbol{y}$ and $f_j(\boldsymbol{x}_q) = f_j(\boldsymbol{y})$, j = 1,…,m.

Step 2.5) Update of EP: Remove from **EP** those solutions dominated by $\boldsymbol{y}$, and add $\boldsymbol{y}$ to **EP** if no solution in **EP** dominates $\boldsymbol{y}$.

Step 3) Stopping condition: If stopping condition is met, then stop and output **EP**. Otherwise, go to **Step 2**.

In Step 2.1, crossover and mutation are applied to $\boldsymbol{x}_k$ and $\boldsymbol{x}_l$ (two neighbors of $\boldsymbol{x}_i$) to generate a new solution $\boldsymbol{y}$. By combining selected portions from two parent solutions, the crossover operator is regarded as the main evolutionary force for offspring production. Offspring solutions inherit some features from their parents. Yet, they are capable of exploring new areas in the search space as long as their parents are not similar to each other.

As mentioned in Section V-A, a solution $\boldsymbol{x} = (SOL(\pi_1),...,SOL(\pi_S))$ is composed of all SOLs associated with all SeNBs in the MEC system, where $SOL(\pi_i) = (SOL(G_{i,1}),...,SOL(G_{i,N_i^{SMD}}))$, i = 1,…,S.

Let $\boldsymbol{x}^{par1}$ and $\boldsymbol{x}^{par2}$ be two parent solutions. Let $SOL^{par1}(G_{i,j}) = (L_{i,j}^{par1}, O_{i,j}^{par1})$ and $SOL^{par2}(G_{i,j}) = (L_{i,j}^{par2}, O_{i,j}^{par2})$ be the SOLs for $G_{i,j}$ in $\boldsymbol{x}^{par1}$ and $\boldsymbol{x}^{par2}$, respectively. The crossover operator is applied to each $SOL^{par1}(G_{i,j})$ and $SOL^{par2}(G_{i,j})$ pair, i = 1,…,S, j = 1,…, N_i^{SMD}, to obtain two offspring SOLs for $G_{i,j}$, namely $SOL^{off1}(G_{i,j})$ and $SOL^{off2}(G_{i,j})$, as described in Algorithm 4, where *randInt*(1, $N_{i,j}$) in step 2 is an integer randomly generated in the range [1, $N_{i,j}$]. To be specific, for pair $SOL^{par1}(G_{i,j})$ and $SOL^{par2}(G_{i,j})$, single-point crossover is applied to the corresponding execution location vector pair, i.e. $L_{i,j}^{par1}$ and $L_{i,j}^{par2}$, and the execution order vector pair, i.e. $O_{i,j}^{par1}$ and $O_{i,j}^{par2}$, respectively.

First, we introduce the execution location (EL) crossover. For each pair ($L_{i,j}^{par1}, L_{i,j}^{par2}$), we randomly generate a crossover point $CPT_{i,j}^{loc}$ (see steps 2-3 in Algorithm 4) and swap the corresponding portions of $L_{i,j}^{par1}$ and $L_{i,j}^{par2}$ before $CPT_{i,j}^{loc}$. Then, we obtain two offspring execution location vectors $L_{i,j}^{off1}$ and $L_{i,j}^{off2}$. According to the task graph in Fig. 2, we present an example of the EL crossover operation in Fig. 7.

In the execution order (EO) crossover, all task precedence constraints must be met. A simple crossover is very likely to produce infeasible execution order vectors for each application, as repetitive tasks may be created. In [20], an effective task execution order crossover operator ensures that all task precedence constraints are always satisfied. This operator is adopted as the EO crossover in MOEA/D-MCOP, as described below.

Algorithm 4. EL and EO Crossovers on Two SOLs Associated with $G_{i,j}$

```
Input: two parent SOLs for $G_{i,j}$, e.g. $SOL^{par1}(G_{i,j}) = (L_{i,j}^{par1}, O_{i,j}^{par1})$ and
    $SOL^{par2}(G_{i,j}) = (L_{i,j}^{par2}, O_{i,j}^{par2})$.
Output: two offspring SOLs for $G_{i,j}$, e.g. $SOL^{off1}(G_{i,j})$ and $SOL^{off2}(G_{i,j})$.
    // the EL crossover
1.  Initialize $L_{i,j}^{off1} = (loc_{i,j,1}^{off1},...,loc_{i,j,N_{i,j}}^{off1})$ and $L_{i,j}^{off2} = (loc_{i,j,1}^{off2},...,loc_{i,j,N_{i,j}}^{off2})$;
2.  Generate a random integer $randInt(1, N_{i,j})$;
3.  Set $CPT_{i,j}^{loc} = randInt(1, N_{i,j})$;
4.  for $k = 1$ to $CPT_{i,j}^{loc}$ do Set $loc_{i,j,k}^{off1} = loc_{i,j,k}^{par2}$ and $loc_{i,j,k}^{off2} = loc_{i,j,k}^{par1}$;
5.  for $k = CPT_{i,j}^{loc} + 1$ to $N_{i,j}$ do Set $loc_{i,j,k}^{off1} = loc_{i,j,k}^{par1}$ and $loc_{i,j,k}^{off2} = loc_{i,j,k}^{par2}$;
    // the EO crossover
6.  Generate a random integer $randInt(1, N_{i,j})$;
7.  Set $CPT_{i,j}^{ord} = randInt(1, N_{i,j})$ and $N_{i,j}^{temp} = CPT_{i,j}^{ord} + N_{i,j}$;
8.  Generate two $N_{i,j}^{temp}$-dimension temporary vectors, e.g.
    $O_{i,j}^{temp1} = (u_{i,j,1}^{temp1},...,u_{i,j,N_{i,j}^{temp}}^{temp1})$ and $O_{i,j}^{temp2} = (u_{i,j,1}^{temp2},...,u_{i,j,N_{i,j}^{temp}}^{temp2})$;
9.  for $k = 1$ to $CPT_{i,j}^{ord}$ do Set $u_{i,j,k}^{temp1} = u_{i,j,k}^{par2}$ and $u_{i,j,k}^{temp2} = u_{i,j,k}^{par1}$;
10. Set $index = 1$;
11. for $k = CPT_{i,j}^{ord} + 1$ to $N_{i,j}^{temp}$ do
12.     Set $u_{i,j,k}^{temp1} = u_{i,j,index}^{par1}$, $u_{i,j,k}^{temp2} = u_{i,j,index}^{par2}$ and $index = index + 1$;
13. Delete repetitive tasks from $O_{i,j}^{temp1}$ and $O_{i,j}^{temp2}$, respectively;
14. Set two offspring EO vectors, $O_{i,j}^{off1} = O_{i,j}^{temp1}$ and $O_{i,j}^{off2} = O_{i,j}^{temp2}$;
    // two offspring SOLs for $G_{i,j}$ are generated
15. Set $SOL^{off1}(G_{i,j}) = (L_{i,j}^{off1}, O_{i,j}^{off1})$ and $SOL^{off2}(G_{i,j}) = (L_{i,j}^{off2}, O_{i,j}^{off2})$.
```

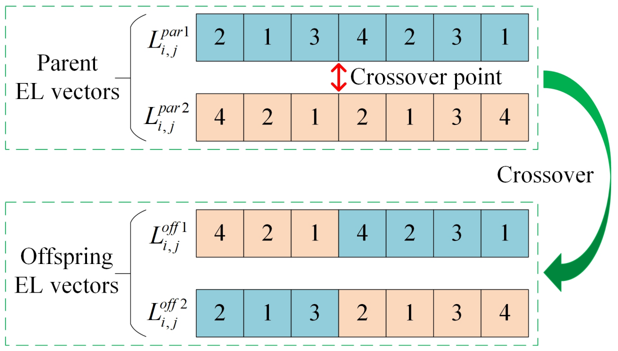


Fig. 7. An example of the EL crossover.

For each pair $(O_{i,j}^{par1}, O_{i,j}^{par2})$, a crossover point $CPT_{i,j}^{ord}$ is randomly selected. Each parent EO vector is divided into two portions by $CPT_{i,j}^{ord}$. The two portions before $CPT_{i,j}^{ord}$ are swapped and then concatenated to $O_{i,j}^{par1}$ and $O_{i,j}^{par2}$, respectively, resulting into two temporary vectors. All repetitive tasks in the temporary vectors are then removed, reserving the order of the remaining tasks, resulting into two feasible execution order vectors, namely $O_{i,j}^{off1}$ and $O_{i,j}^{off2}$. An example of the EO crossover operation applied to the task graph in Fig. 2 is shown in Fig. 8.

Based on Algorithm 4, we design the crossover operator on two parent solutions in Algorithm 5.

Mutation plays an important role in introducing diversity to the evolution. Bitwise mutation is applied to the execution location vector $L_{i,j}^{par}$, and single-point mutation is applied to the execution order vector $O_{i,j}^{par}$ in each $SOL^{par}(G_{i,j})$, $i = 1,\ldots,S$, $j = 1,\ldots, N_i^{SMD}$, to obtain an offspring SOL for $G_{i,j}$, namely $SOL^{off}(G_{i,j})$. This is described in Algorithm 6, where $random(0,1)$ is a number randomly generated in the range (0, 1). A mutation probability for all applications in the MEC system, MP^{app}, is used to decide if $SOL(G_{i,j})$ is mutated, $i = 1,\ldots,S, j = 1,\ldots, N_i^{SMD}$.

In the EL mutation, a mutation probability for $L_{i,j}^{par}$, $MP_{i,j}^{loc}$, is adopted to decide if each execution location in $L_{i,j}^{par}$ is mutated. If an execution location $loc_{i,j,k}^{par}$ in $L_{i,j}^{par}$ is chosen for mutation, a random integer number from $\{1,\ldots,H+1\}$ is used to replace $loc_{i,j,k}^{par}$. After mutation, an execution location vector $L_{i,j}^{off}$ is generated. For the task graph in Fig. 2, an example of the EL mutation operation is presented in Fig. 9.

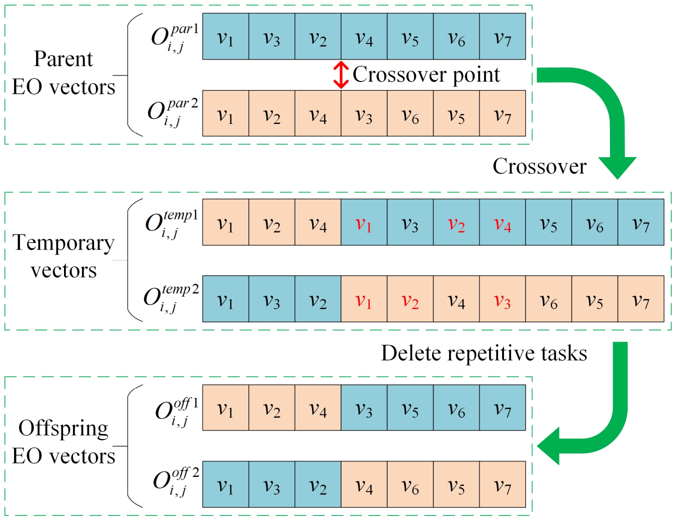

Parent EO vectors
$O_{i,j}^{par1}$
v_1 v_3 v_2 v_4 v_5 v_6 v_7
Crossover point
$O_{i,j}^{par2}$
v_1 v_2 v_4 v_3 v_6 v_5 v_7
Crossover
Temporary vectors
$O_{i,j}^{temp1}$
v_1 v_2 v_4 v_1 v_3 v_2 v_4 v_5 v_6 v_7
$O_{i,j}^{temp2}$
v_1 v_3 v_2 v_1 v_2 v_4 v_3 v_6 v_5 v_7
Delete repetitive tasks
Offspring EO vectors
$O_{i,j}^{off1}$
v_1 v_2 v_4 v_3 v_5 v_6 v_7
$O_{i,j}^{off2}$
v_1 v_3 v_2 v_4 v_6 v_5 v_7

Fig. 8. An example of the EO crossover applied to the task graph in Fig. 2.

Algorithm 5. Crossover Procedure on Two Solutions

```
Input: two parent solutions, e.g. $x^{par1}$ and $x^{par2}$.
Output: two offspring solutions, e.g. $x^{off1}$ and $x^{off2}$.
1.  Initialize $x^{off1} = (SOL^{off1}(\pi_1),\ldots,SOL^{off1}(\pi_S))$ and
    $x^{off2} = (SOL^{off2}(\pi_1),\ldots,SOL^{off2}(\pi_S))$;
    // for each $SOL^{par1}(\pi_i)$ and $SOL^{par2}(\pi_i)$ pair
2.  for $i = 1$ to $S$ do
        // for each $SOL^{par1}(G_{i,j})$ and $SOL^{par2}(G_{i,j})$ pair
3.      for $j = 1$ to $N_i^{SMD}$ do
4.          Obtain two offspring SOLs, $SOL^{off1}(G_{i,j})$ and $SOL^{off2}(G_{i,j})$,
            by running Algorithm 4 on $SOL^{par1}(G_{i,j})$ and $SOL^{par2}(G_{i,j})$;
```

In the proposed MOEA/D, the EO mutation in [20] is adopted, where all task precedence constraints are met. Let $O_{i,j}^{par} = (u_{i,j,1}^{par},\ldots,u_{i,j,k}^{par},\ldots,u_{i,j,N_{i,j}}^{par})$ be the parent execution order vector associated with $G_{i,j}$ chosen for mutation. $u_{i,j,k}^{par}$ is the

k-th task to be executed in $O_{i,j}^{par}$, $u_{i,j,k}^{par} \in V_{i,j}$, $u_{i,j,1}^{par} = v_{i,j,start}$, and $u_{i,j,N_{i,j}}^{par} = v_{i,j,end}$.

Algorithm 6. EL and EO Mutation Procedures

```
Input: parent SOL for $G_{i,j}$, e.g. $SOL^{par}(G_{i,j}) = (L_{i,j}^{par}, O_{i,j}^{par})$.
Output: offspring SOL for $G_{i,j}$, e.g. $SOL^{off}(G_{i,j})$.
1.  Generate a random number $random(0, 1)$;
2.  Set $MP_{rnd}^{app} = random(0, 1)$;
3.  if $MP_{rnd}^{app} \le MP^{app}$ then            // $SOL^{par}(G_{i,j})$ is to be mutated
        // the EL mutation
4.      Initialize the offspring EL vector $L_{i,j}^{off} = (loc_{i,j,1}^{off},...,loc_{i,j,N_{i,j}}^{off})$;
5.      for $k = 1$ to $N_{i,j}$ do
6.          Generate a random number $random(0, 1)$;
7.          Set $MP_{rnd}^{loc} = random(0, 1)$;
8.          if $MP_{rnd}^{loc} \le MP_{i,j}^{loc}$ then
9.              Generate a random integer $randInt(1, H+1)$;
10.             Set $loc_{i,j,k}^{off} = randInt(1, H+1)$;
11.         else Set $loc_{i,j,k}^{off} = loc_{i,j,k}^{par}$;
        // the EO mutation
12.     Initialize the offspring EO vector $O_{i,j}^{off} = (u_{i,j,1}^{off},...,u_{i,j,N_{i,j}}^{off})$;
13.     Randomly select a task $u_{i,j,r}^{par}$ from $O_{i,j}^{par}$, where $u_{i,j,r}^{par}$
        cannot be the start task nor the end task;
14.     Carry out forward search until the last predecessor of $u_{i,j,r}^{par}$,
        e.g. $u_{i,j,a}^{par}$, is found;
15.     Include the visited tasks in vector $(u_{i,j,1}^{par},...,u_{i,j,a}^{par})$;
16.     Carry out backward search until the last successor of $u_{i,j,r}^{par}$,
        e.g. $u_{i,j,b}^{par}$, is found;
17.     Include the visited tasks in vector $(u_{i,j,b}^{par},...,u_{i,j,N_{i,j}}^{par})$;
18.     Generate a temporary vector $temp = (u_{i,j,a+1}^{par},...,u_{i,j,b-1}^{par})$;
19.     Randomly select a location in $temp$ and move $u_{i,j,r}^{par}$ there;
20.     Set $O_{i,j}^{off} = (u_{i,j,1}^{par},...,u_{i,j,a}^{par}) \circ temp \circ (u_{i,j,b}^{par},...,u_{i,j,N_{i,j}}^{par})$;
21.     Set $SOL^{off}(G_{i,j}) = (L_{i,j}^{off}, O_{i,j}^{off})$;
22. else                                          // $SOL^{par}(G_{i,j})$ is not mutated
23.     Set $SOL^{off}(G_{i,j}) = SOL^{par}(G_{i,j})$.
```

A task in $O_{i,j}^{par}$, $u_{i,j,r}^{par}$, is first randomly selected, where $r \in \{2,\ldots,N_{i,j}-1\}$. Task $u_{i,j,r}^{par}$ can be neither the start task nor the end task. Next, a forward search from $u_{i,j,1}^{par}$ to the last predecessor of $u_{i,j,r}^{par}$, e.g. $u_{i,j,a}^{par}$, is implemented on $O_{i,j}^{par}$. The visited tasks are included in a vector $(u_{i,j,1}^{par},...,u_{i,j,a}^{par})$. Similarly, a backward search from $u_{i,j,N_{i,j}}^{par}$ is implemented on $O_{i,j}^{par}$. When the last successor of $u_{i,j,r}^{par}$, e.g. $u_{i,j,b}^{par}$, is found, the search terminates. The visited tasks are included in a vector $(u_{i,j,b}^{par},...,u_{i,j,N_{i,j}}^{par})$. Let $temp = (u_{i,j,a+1}^{par},...,u_{i,j,b-1}^{par})$ denote a temporary vector between the two vectors of visited tasks. A location other than the current location of $u_{i,j,r}^{par}$ is randomly selected in *temp*, where $u_{i,j,r}^{par}$ is then moved to.

The EO mutation produces $O_{i,j}^{off} = (u_{i,j,1}^{par},...,u_{i,j,a}^{par}) \circ temp \circ (u_{i,j,b}^{par},...,u_{i,j,N_{i,j}}^{par})$, where "∘" concatenates the above produced three vectors. An example of the EO mutation operation applied to the task graph in Fig. 2 is illustrated in Fig. 10.

Based on Algorithm 6, Algorithm 7 presents how a parent solution is mutated.

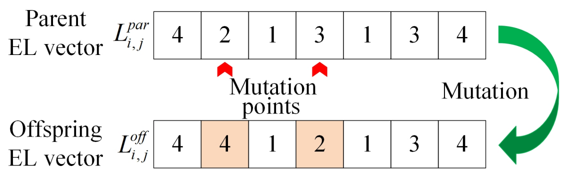


Fig. 9. An example of the EL mutation.

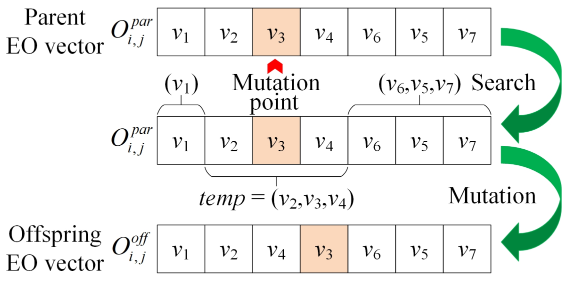


Fig. 10. An example of the EO mutation applied to the task graph in Fig. 2.

Algorithm 7. Mutation Procedure on One Solution

```
Input: parent solution, e.g. $x^{par}$.
Output: offspring solution, e.g. $x^{off}$.
1. Initialize $x^{off} = (SOL^{off}(\pi_1),\ldots,SOL^{off}(\pi_S))$;
2. for $i = 1$ to $S$ do                          // for each $SOL^{par}(\pi_i)$
3.     for $j = 1$ to $N_i^{SMD}$ do              // for each $SOL^{par}(G_{i,j})$
4.         Obtain the offspring SOL $SOL^{off}(G_{i,j})$ by running
           Algorithm 6 on $SOL^{par}(G_{i,j})$;
```

E. Complexity Analysis

Let $O(f)$ be the time complexity for evaluating a solution to the MCOP. Let N_{total}^{task} and M be the total number of tasks in the MEC system and the number of the computing frequency levels on a core, respectively. Let the number of objectives and that of neighbors for subproblem represented by m and W, respectively. Let the size of the external population (**EP**) denoted by |**EP**|.

First, we analyze the complexity of each step in the loop. As the encoding length of each solution is N_{total}^{task}, Step 2.1 (simple crossover and mutation) has a time complexity of $O(N_{total}^{task})$. There are two operations in Step 2.2 (namely the DVFS-EC scheme), including solution evaluation and solution improvement. In the first part, $O(f)$ is the time complexity as defined above. In the second part, a solution is improved in terms of energy consumption. The DVFS technique is applied to each locally executed task, resulting into a time complexity of $O(M)$. In the worst case, all tasks are executed on SMDs, which corresponds to a time complexity of $O(N_{total}^{task} \cdot M)$. Hence, Step 2.2 has a time complexity of $O(f + N_{total}^{task} \cdot M)$. Steps 2.3, 2.4 and 2.5 update the reference point, the neighbors of each subproblem and **EP**, leading to time complexities of $O(m)$, $O(m \cdot W)$ and $O(m \cdot |\mathbf{EP}|^2)$, respectively. As the MCOP problem is highly complicated, solution evaluation is the most time-consuming operation in the loop. Compared with it, other steps are trivial. Hence, the time complexity of the loop (Step 2) is reduced to $O(f)$.

Then, we analyze the complexity of MOEA/D-MCOP. Compared with the loop, the time complexity of the initialization is trivial and thus can be ignored. Hence, MOEA/D-MCOP is only dependent on the complexity of the loop, the number of subproblems, N_p, and the predefined number of iterations, G_{max}, leading to a time complexity of $O(f \cdot N_p \cdot G_{max})$.

VI. Experiments and Results Analysis

A. Test Instances

In this paper, we consider a centralized MEC system with a radius of 100 m. The network parameter setup method in [30] is adopted. A system with five small cells is regarded as a medium-scale MEC scenario, which meets most of the users' requirements. Therefore, we also use the five-small-cell MEC network to conduct all experiments. The cells are evenly scattered, each with a radius of 50 m. The number of channels in the MEC system is fixed to 10 for simplicity purpose. To guarantee there is no channel interference between SMDs within any small cell, we randomly generate the number of SMDs in each small cell in the range of [3, 9].

For an arbitrary SMD $U_{i,j}$, the maximum computing frequency of the 1st core, $f_{i,j,1}^{max}$, is randomly generated in the range [0.5, 1] GHz. The maximum computing frequencies of the 2nd and 3rd cores are set to $f_{i,j,2}^{max} = f_{i,j,1}^{max} - 0.1$ and $f_{i,j,3}^{max} = f_{i,j,1}^{max} - 0.25$, respectively. Assume the power consumption of the 1st, 2nd, and 3rd cores at the maximum computing frequency is 4 W, 2 W, and 1 W, respectively. According to [25], we assume each core has 4 computing frequency levels with scaling factors $\alpha_1 = 0.2$, $\alpha_2 = 0.5$, $\alpha_3 = 0.8$, and $\alpha_4 = 1$, respectively, and constant γ is set to 2. The problem characteristics are shown in Table III.

For the application generation, we first randomly generate the number of tasks, and then randomly generate their data size and the number of CPU cycles required to perform them. The task-precedence constraints between tasks are randomly generated based on the task generation method introduced in [20]. To be specific, we randomly generated the number of tasks in an application using six different ranges to control the scale of the MCOP problem, as shown in Table IV. In each instance, the number of tasks is randomly generated according to the corresponding range.

TABLE III
Problem Characteristics

Parameter	Value
Number of SeNBs in the MEC system (S)	5
Total bandwidth offered by the MEC system (B_{total})	20 MHz
Number of channels offered by the MEC system ($N_{channel}$)	10
Noise power (σ^2)	−176 dBm
Computing capability of the MEC server (F)	4 GHz
Number of heterogeneous cores of a SMD (H)	3
Number of the computing frequency levels on a core (M)	4
Number of SMDs in the i-th small cell (N_i^{SMD})	3-9
Power consumption when $U_{i,j}$ offloads tasks ($p_{i,j}^{txd}$)	0.5 W
Power consumption when $U_{i,j}$ receives data ($p_{i,j}^{rxd}$)	0.1 W
Number of CPU cycles required to perform $v_{i,j,k}$ ($c_{i,j,k}$)	[0.1, 0.5] GHz
Input data size of $v_{i,j,k}$ ($d_{i,j,k}$)	[5000, 6000] Kb
Output data size of $v_{i,j,k}$ ($o_{i,j,k}$)	[500, 1000] Kb

TABLE IV
Six Test Instances

Instance No.	Number of tasks in an application	Total number of tasks in the MEC system
1	10-20	349
2	15-25	566
3	20-30	647
4	25-35	948
5	30-40	1230
6	10-40	630

B. Experiment Setup

We ran the experiments on a computer with Windows 10 OS, Intel(R) Core(TM) i7-8700 CPU 3.2 GHz and 16 GB RAM. All algorithms were implemented using Python 3.6. The parameters of the proposed MOEA/D-MCOP are listed in Table V. The results are obtained by running each algorithm 20 times, from which the statistics are collected and analyzed.

TABLE V
Parameters of MOEA/D-MCOP

Parameter	Value
Population size (N_p)	100
Predefined number of iterations	100
Number of neighbors for each subproblem (W)	10
Probability for mutating a SOL associated with $G_{i,j}$ (MP^{app})	$1/N_{total}^{SMD}$
Probability for mutating an EL associated with $G_{i,j}$ ($MP_{i,j}^{loc}$)	$1/N_{i,j}$

C. Performance Measures

Four widely recognized performance metrics in the literature [31][32] are used to thoroughly evaluate the performance of MOEA/D-MCOP. Let PF_{ref} and PF_{known} denote the reference PF approximating the true PF and the PF obtained by an algorithm, respectively.

For the new MCOP problem concerned in this paper, the true PF is not known. A widely used method in the research is to collect the best so far solutions found by all algorithms in all runs and obtain the PF associated with those nondominated ones as PF_{ref}.

- Inverted generational distance (IGD)

IGD as defined in Eq. (21) can simultaneously measure the convergence and diversity of a given PF. For an algorithm, a smaller IGD value reflects better overall performance.

$$IGD = \frac{\sum_{\tau_{ref} \in PF_{ref}} d(\tau_{ref}, PF_{known})}{\left|PF_{ref}\right|} \tag{21}$$

where $|PF_{ref}|$ is the number of points in PF_{ref} and $d(\tau_{ref}, PF_{known})$ is the Euclidean distance between point τ_{ref} in PF_{ref} and its nearest point in PF_{known}.

- Generational distance (GD)

GD as defined in Eq. (22) can measure how closely PF_{known} converges to PF_{ref}.

$$GD = \sqrt{\frac{\sum_{\tau_{known} \in PF_{known}} d(\tau_{known}, PF_{ref})}{\left|PF_{known}\right|}} \tag{22}$$

where $|PF_{known}|$ is the number of points in PF_{known} and $d(\tau_{known}, PF_{ref})$ is the Euclidean distance between point τ_{known} in PF_{known} and its nearest point in PF_{ref}.

- Student's *t*-test

In this paper, two-tailed *t*-test with 38 degrees of freedom at a 0.05 level of significance [49] is utilized to compare two algorithms Alg.1 and Alg.2 based on the IGD values obtained in 20 runs. The results show whether performance of Alg.1 is significantly better than, significantly worse than, or statistically equivalent to that of Alg.2, respectively.

- Friedman test

The Friedman test [50] is a non-parametric test for detecting the differences among different algorithms in terms of IGD and GD. All algorithms under comparisons are ranked and their average ranks explicitly indicate how well they perform.

D. *Effectiveness of Two Performance Enhancing Schemes*

To demonstrate the effectiveness of the two new schemes, namely PSPI in Section V-B and DVFS-EC in Section V-C, in the proposed MOEA/D-MCOP, the following three variants of MOEA/D are tested on the six test instances in Table IV.

- MOEA/D: the original MOEA/D [7].
- MOEA/D-PSPI: MOEA/D with the PSPI scheme.
- MOEA/D-MCOP: MOEA/D-PSPI with the DVFS-EC scheme.

For the three algorithms above, the population size and the predefined number of iterations are set to 100, respectively.

The results of *mean* and standard deviation (SD) of IGD and GD are collected in Tables VI and VII, respectively. It is obvious that MOEA/D-PSPI outperforms MOEA/D against the two performance measures in all instances. This is due to that the problem specific knowledge incorporated in the PSPI scheme is able to guide the search to start from promising areas. Moreover, MOEA/D-MCOP achieves better mean value than the other two in terms of IGD and GD in each instance. This shows that the DVFS-EC scheme helps to reduce the energy consumption of SMDs without sacrificing completion time, thus enhances the local exploitation ability of the search.

Fig. 11 shows the PFs obtained by the three algorithms. It can be seen clearly that both the PSPI and DVFS-EC schemes contribute to performance improvement of MOEA/D.

TABLE VI
RESULTS OF IGD (BEST RESULTS ARE IN BOLD)

Instance No.	MOEA/D	MOEA/D-PSPI	MOEA/D-MCOP
1	7.86(0.16)	7.64(0.15)	**0.19**(0.21)
2	4.88(0.25)	4.19(0.11)	**0.47**(0.14)
3	3.14(0.16)	3.11(0.24)	**0.81**(0.29)
4	8.91(0.39)	7.06(0.19)	**1.14**(0.30)
5	11.60(0.39)	11.23(0.25)	**1.68**(0.55)
6	5.74(0.22)	5.48(0.16)	**0.55**(0.21)

TABLE VII
RESULTS OF GD (BEST RESULTS ARE IN BOLD)

Instance No.	MOEA/D	MOEA/D-PSPI	MOEA/D-MCOP
1	2.84(0.04)	2.74(0.03)	**0.31**(0.29)
2	1.73(0.04)	1.58(0.03)	**0.56**(0.10)
3	1.15(0.04)	1.09(0.04)	**1.03**(0.15)
4	2.00(0.08)	1.69(0.03)	**0.89**(0.26)
5	1.90(0.06)	1.81(0.04)	**1.63**(0.17)
6	1.70(0.04)	1.60(0.02)	**0.52**(0.11)

E. *Overall Performance Evaluation*

MOEA/D-MCOP is compared against the following eight state-of-the-art algorithms, i.e. five MOEAs and three heuristic algorithms, in six test instances in Table IV.

- NSGA-II: the modified fast and elitist nondominated sorting genetic algorithm [30] used to achieve a trade-off between the average energy consumption and the average completion time in a MEC network.
- MOWOA: the multiobjective whale optimization algorithm [27] applied to address the multiobjective task workflow scheduling problem, where weighted sum is used to aggregate workflow completion time and energy consumption into one objective function.
- MOFOA: the knowledge-guided multiobjective fruit fly optimization algorithm [51] developed to tackle the multi-skill resource-constrained project scheduling problem, where the completion time and the total cost are minimized at the same time.
- HGPCA: the hierarchical GA and PSO-based computation algorithm [23] proposed to solve the multi-user offloading game problem in MCC, where the energy consumption of SMDs is minimized.
- MOEA/D: the multiobjective evolutionary algorithm based on decomposition [7] with the Tchebycheff method.

- TSDVFS: the task scheduling with dynamic voltage and frequency scaling algorithm [25] developed to minimize the energy consumption of SMDs in MCC, where the application completion time constraint and the task-precedence constraints are satisfied.
- CTESA: the collaborative task execution scheduling algorithm [22] devised to address the delay-constrained workflow scheduling problem in MCC network. CTESA minimizes the energy consumption of SMD(s) while meeting the application completion time deadline.
- eDors: the energy-efficient dynamic offloading and resource scheduling algorithm [28] presented to reduce the energy consumption and shorten the application completion time, where the task-dependency requirement and application completion time deadline are constrained.
- MOEA/D-MCOP: the proposed MOEA/D with the PSPI and DVFS-EC schemes in this paper.

For all MOEAs under comparison, the population size and the predefined number of iterations are set to 100, respectively. To make a fair comparison, we directly adopt the parameter settings in NSGA-II [30], MOWOA [27], MOFOA [51], HGPCA [23], and MOEA/D [7]. To be specific, in NSGA-II, the crossover and mutation probabilities are set to 0.8 and 0.3, respectively. In MOWOA, the upper and lower bounds of the search range are set to 4.4 and 0.5, respectively. For MOFOA, we set the sub-swarm size, the learning rate of the experience, and the number of elite fruit flies to 5, 0.1, and 3, respectively. In HGPCA, the crossover probability, the mutation probability, the inertia weight, and the acceleration instant are set to 0.6, 0.01, 0.4, and 1.5, respectively. In MOEA/D and MOEA/D-MCOP, the number of neighbors for each subproblem is set to 10.

Note that each of three heuristics only obtains a single solution after each run. To make a fair comparison, each heuristic should obtain a set of nondominated solutions for performance comparison. Hence, we repeatedly run a heuristic with incrementally increased application completion time deadline as a constraint. Each deadline results into a solution with explicit application completion time and energy consumption. By doing so, each heuristic can obtain a set of nondominated solutions after a number of runs.

We first compare the average completion time of applications, i.e. ACT, and the average energy consumption of SMDs, i.e. AEC, obtained by the nine algorithms. Figs. 12 and 13 depict the box plots of the nine algorithms in terms of ACT and AEC, respectively. In Fig. 12, one can observe that MOEA/D-MCOP performs better than the other eight algorithms in most of the test instances (except Instances 3 and 4). This is because the PSPI scheme in MOEA/D-MCOP adopts the latency-based execution location initialization method (LELI). By reducing the completion time of each task in a greedy manner, LELI can reduce the completion time of each application, which also helps to reduce the ACT in the MEC system.

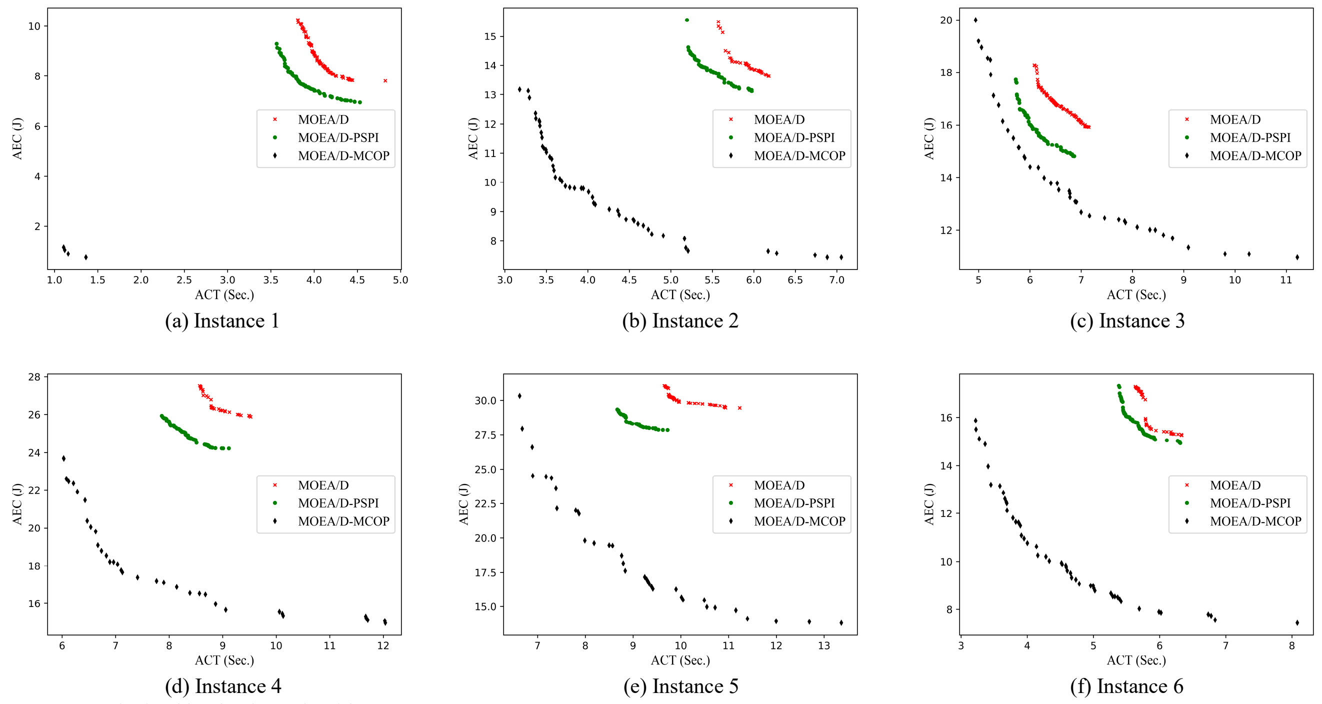
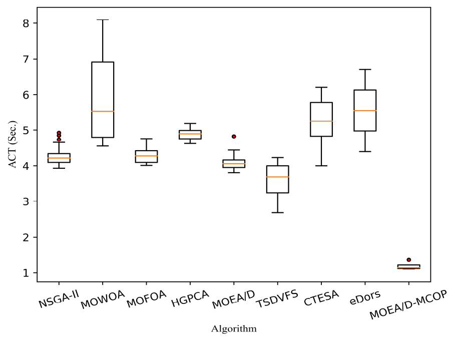
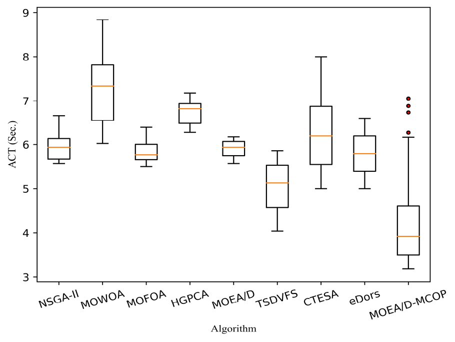


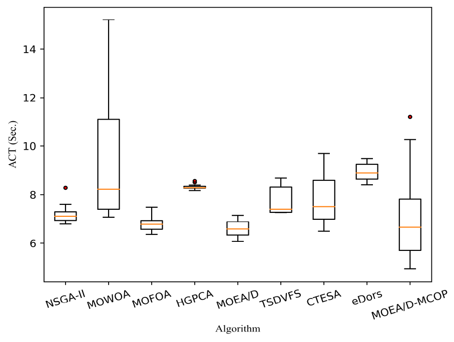
Fig. 11. PFs obtained by the three algorithms.



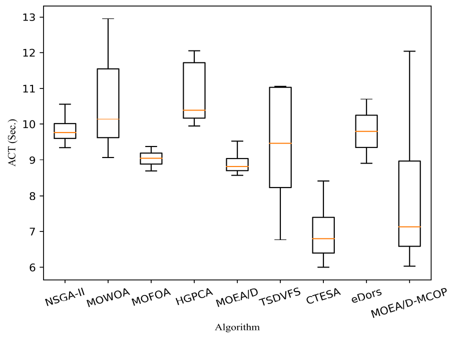
(a) Instance 1



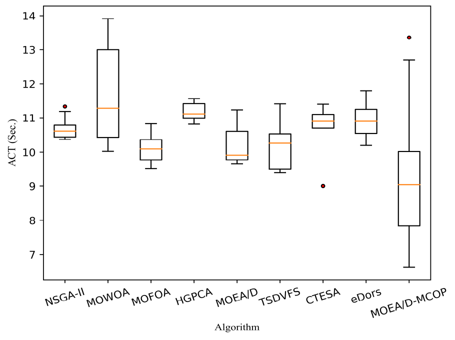
(b) Instance 2



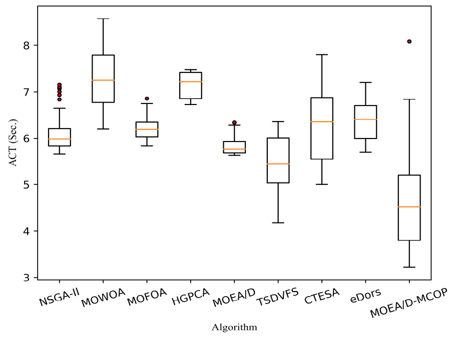
(c) Instance 3



(d) Instance 4

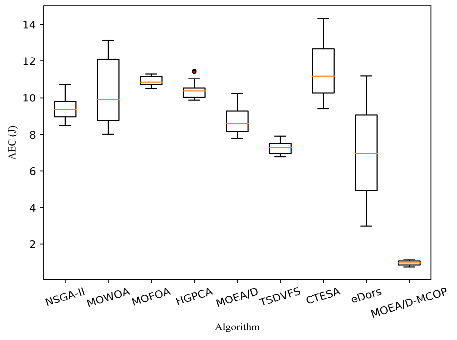


(e) Instance 5

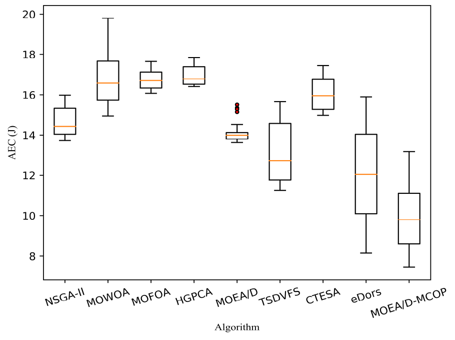


(f) Instance 6

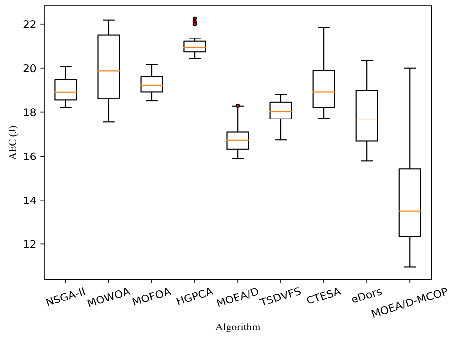
Fig. 12. Box plots of the nine algorithms in terms of ACT.



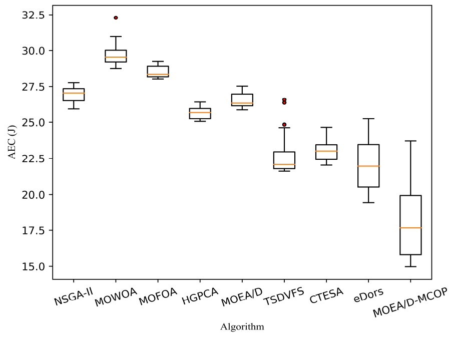
(a) Instance 1



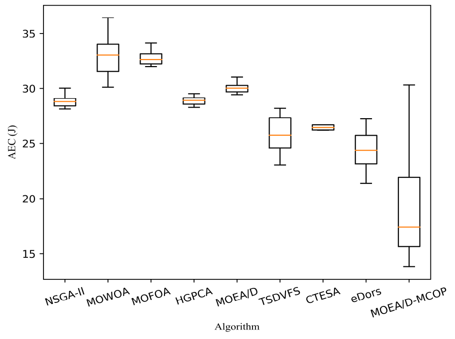
(b) Instance 2



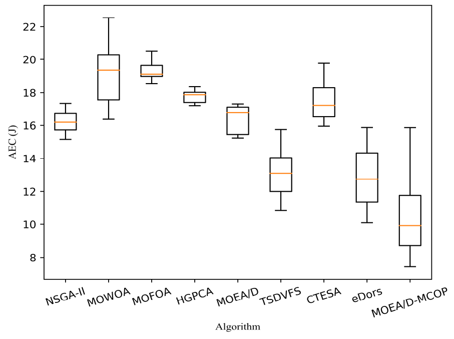
(c) Instance 3



(d) Instance 4



(e) Instance 5



(f) Instance 6

Fig. 13. Box plots of the nine algorithms in terms of AEC.

In Fig. 13, there is no doubt MOEA/D-MCOP is the best. This is because the DVFS-EC scheme can significantly reduce the AEC by dynamically adjusting the frequency level of each core. Besides, eDors and TSDVFS are the second- and third-best algorithms, respectively. The two algorithms decrease the energy consumption of SMDs thanks to DVFS. Meanwhile, eDors always overweighs TSDVFS with respect to the AEC. The reason behind it is that eDors takes advantage of transmission power control mechanism, which further reduces the energy consumption. However, both eDors and TSDVFS are not good at obtaining a decent tradeoff between ACT and AEC, i.e. improving one objective harms the other. On the other hand, if we consider ACT and AEC together, MOEA/D-MCOP achieves the best overall performance.

The mean and SD values of IGD and GD obtained by all algorithms are shown in Tables VIII and IX, respectively.

Firstly, if taking all algorithms into account, one can observe that MOEA/D-MCOP performs the best with respect to IGD and GD in all instances. IGD reflects the diversification and convergence of nondominated solutions simultaneously. GD reveals how far the obtained PF is from the reference PF. Results in Tables VIII and IX show that MOEA/D-MCOP always obtains the set of nondominated solutions closest to the reference PF, which indicates MOEA/D-MCOP achieves a better trade-off between global exploration and local exploitation.

Secondly, it is easily seen that MOEA/D outperforms all MOEAs except MOEA/D-MCOP in almost all instances, showing that MOEA/D is highly effective for the MCOP problem. On the one hand, NSGA-II, MOWOA, MOFOA and HGPCA are Pareto-dominance based. If parameters are not set appropriately, they are likely to get stuck into local optima and converge slowly. On the other hand, MOEA/D is decomposition-based, addressing a number of scalar optimization subproblems in parallel. Compared with those Pareto-dominance based MOEAs, MOEA/D is featured with stronger global exploration capability. Therefore, MOEA/D performs better than NSGA-II, MOWOA, MOFOA, and HGPCA. This also justifies why MOEA/D is chosen for addressing the MCOP problem.

Thirdly, among heuristics, TSDVFS is the winner as it outperforms CTESA and eDors in most test instances except Instances 3 and 4 in terms of IGD and GD. TSDVFS first adopts the initial scheduling algorithm to generate the minimal-delay schedule. Then, it applies DVFS technology to reduce the energy consumption of SMDs. However, TSDVFS cannot strike a balance between the application completion time and the energy consumption of SMDs. This is why TSDVFS is beaten by MOEA/D-MCOP. On the other hand, eDors and CTESA also have obvious drawbacks. eDors is not good at reducing of the application completion time. CTESA schedules the tasks on the partial critical path rather than considering the task graph as a whole.

TABLE VIII
RESULTS OF IGD (BEST RESULTS ARE IN BOLD)

Algorithm	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6
NSGA-II	8.71(0.24)	5.37(0.29)	5.07(0.24)	9.22(0.41)	10.77(0.44)	5.93(0.19)
MOWOA	9.94(0.16)	7.21(0.25)	6.42(0.32)	11.88(0.31)	13.32(0.47)	8.24(0.34)
MOFOA	10.39(0.16)	6.97(0.20)	5.15(0.16)	10.40(0.23)	13.80(0.24)	8.93(0.26)
HGPCA	10.47(0.32)	7.92(0.35)	8.29(0.54)	8.66(0.39)	11.07(0.52)	8.45(0.34)
MOEA/D	7.86(0.16)	4.88(0.25)	3.14(0.16)	8.91(0.39)	11.60(0.39)	5.74(0.22)
TSDVFS	6.40(0)	2.78(0)	3.64(0)	4.45(0)	5.79(0)	2.22(0)
CTESA	9.80(0)	6.14(0)	4.63(0)	4.16(0)	7.99(0)	6.51(0)
eDors	8.33(0)	3.93(0)	3.22(0)	3.56(0)	10.33(0)	8.25(0)
MOEA/D-MCOP	**0.19**(0.21)	**0.47**(0.14)	**0.81**(0.29)	**1.14**(0.30)	**1.68**(0.55)	**0.55**(0.21)

TABLE IX
RESULTS OF GD (BEST RESULTS ARE IN BOLD)

Algorithm	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6
NSGA-II	2.99(0.05)	1.84(0.05)	1.53(0.07)	2.25(0.10)	2.12(0.08)	1.75(0.04)
MOWOA	3.30(0.03)	2.44(0.04)	2.07(0.11)	2.77(0.10)	2.73(0.15)	2.49(0.07)
MOFOA	3.24(0.03)	2.14(0.04)	1.36(0.03)	2.30(0.04)	2.12(0.03)	2.18(0.04)
HGPCA	3.24(0.05)	2.36(0.07)	2.14(0.10)	2.28(0.13)	2.31(0.11)	2.21(0.07)
MOEA/D	2.84(0.04)	1.73(0.04)	1.15(0.04)	2.00(0.08)	1.90(0.06)	1.70(0.04)
TSDVFS	2.56(0)	1.21(0)	1.17(0)	1.78(0)	1.78(0)	1.34(0)
CTESA	3.32(0)	2.09(0)	1.70(0)	0.91(0)	1.93(0)	1.93(0)
eDors	3.47(0)	2.56(0)	2.46(0)	3.25(0)	2.17(0)	2.46(0)
MOEA/D-MCOP	**0.31**(0.29)	**0.56**(0.10)	**1.03**(0.15)	**0.89**(0.26)	**1.63**(0.17)	**0.52**(0.11)

The results of Student's *t*-test based on IGD are shown in Table X. MOEA/D-MCOP is clearly the best among all algorithms. Friedman test is also utilized to rank algorithm performance. Based on the IGD and GD values, the average rankings of the nine algorithms are shown in Table XI. The PFs obtained by the nine algorithms are shown in Fig. 14. Table XI and Fig. 14 both demonstrate the superiority of MOEA/D-MCOP over the rest of the algorithms.

TABLE X

RESULTS OF STUDENT'S *T*-TEST BASED ON IGD

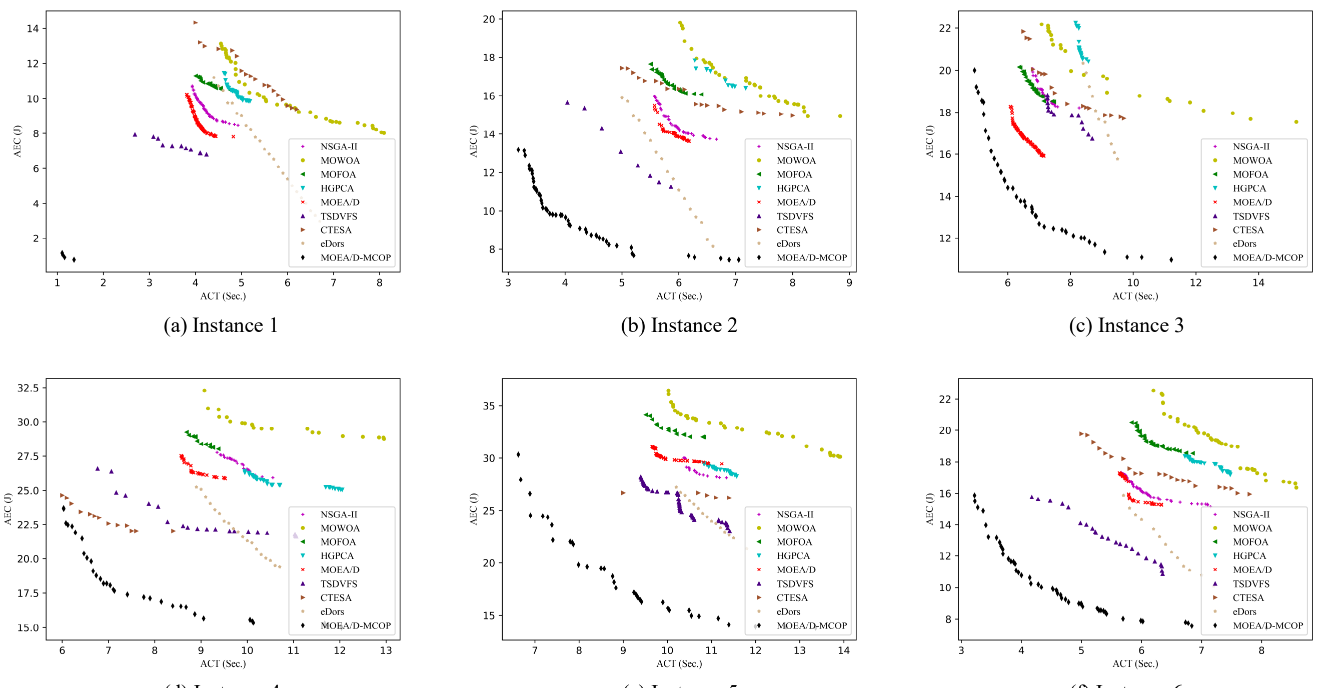
Alg.1 ↔ Alg.2	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6
MOEA/D-MCOP ↔ NSGA-II	+	+	+	+	+	+
MOEA/D-MCOP ↔ MOWOA	+	+	+	+	+	+
MOEA/D-MCOP ↔ MOFOA	+	+	+	+	+	+
MOEA/D-MCOP ↔HGPCA	+	+	+	+	+	+
MOEA/D-MCOP ↔ MOEA/D	+	+	+	+	+	+
MOEA/D-MCOP ↔ TSDVFS	+	+	+	+	+	+
MOEA/D-MCOP ↔ CTESA	+	+	+	+	+	+
MOEA/D-MCOP ↔ eDors	+	+	+	+	+	+

Symbol "+" indicates Alg.1 performs significantly better than Alg.2.

TABLE XI

RANKINGS OF ALL ALGORITHMS ON IGD AND GD

Algorithm	IGD		GD	
	Average rank	Position	Average rank	Position
NSGA-II	5.33	6	4.50	4
MOWOA	7.67	7	8.00	8
MOFOA	8.00	9	5.67	6
HGPCA	7.67	8	7.00	7
MOEA/D	4.17	4	3.00	3
TSDVFS	2.67	2	2.33	2
CTESA	4.67	5	5.00	5
eDors	3.83	3	8.50	9
MOEA/D-MCOP	1.00	1	1.00	1



(a) Instance 1 (b) Instance 2 (c) Instance 3

(d) Instance 4 (e) Instance 5 (f) Instance 6

Fig. 14. PFs obtained by the nine algorithms.

VII. CONCLUSIONS AND FUTURE WORK

A. Conclusions

This paper models a new multiobjective computation offloading problem (MCOP) in mobile edge computing (MEC) environment, where two objectives, namely the average completion time of applications and the average energy consumption of all smart mobile devices (SMDs), are minimized simultaneously. This new MCOP model, for the first time, considers the task-precedence constraints within each application in MEC, where an ordered list of tasks should be executed one by one.

To address the new problem, an improved MOEA/D with two extensions, namely MOEA/D-MCOP is proposed. The first extension is a problem-specific population initialization scheme that generates high-quality initial population. The second extension is a DVFS-based energy conservation scheme that improves the quality of a given solution by reducing the energy consumption of SMDs. Simulation results demonstrate that the proposed MOEA/D-MCOP performs better than the five state-of-the-art MOEAs and three heuristics in terms of the average completion time, the average energy consumption, inverted generational distance, generational distance, *t*-test, and Friedman test.

B. *Future Work*

The MCOP problem modeled in this paper is a static optimization problem in MEC network, where the number of SMDs remains unchanged and SMDs do not move during computation offloading. However, in the real world, dynamic and uncertainty are key features in MEC networks, such as mobility, ever-changing wireless channel and number of SMDs. We will study the MCOP problem in a dynamic MEC environment, taking the three issues above into consideration. In this case, MOEA/D-MCOP cannot respond within a short time to the dynamic MEC network, especially when SMDs move quickly. Therefore, we will concentrate on developing online algorithms and models in future work, e.g. problem-specific heuristics, and deep reinforcement learning based models.

The computing resources on MEC servers and the spectrum resources in wireless channels are both limited in MEC environment. Therefore, it is of significance to study how the computing and spectrum resources are reasonably allocated between SMDs in MEC networks. Moreover, we will jointly consider computation offloading, resource allocation, content caching, and task-precedence constraints among tasks to meet the requirements of various applications. To be specific, we will study a centralized MEC scenario with limited computing and spectrum resources, jointly taking computation offloading, resource allocation, content caching, and task-precedence constraints into account. We will model this complicated scenario as a new multiobjective optimization problem (MOP). There are three objectives for minimization at the same time, including the completion time of applications, the energy consumption of SMDs and the resource cost of SMDs. The resource cost includes the cost for renting computing resources from MEC servers, and that for leasing spectrum resources from small cell. In addition, we will propose an efficient multiobjective optimization algorithm to address the problem.

References

[1] E. Novak, Z. Tang, and Q. Li, "Ultrasound proximity networking on smart mobile devices for IoT applications," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 399–409, Feb. 2019.

[2] D. Mazza, A. Bernaus, D. Tarchi, A. A. Juan, and G. Corazza, "Supporting mobile cloud computing in smart cities via randomized algorithms," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1598–1609, Jun. 2018.

[3] D. Yao, C. Yu, L. Yang, and H. Jin, "Using crowdsourcing to provide QoS for mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 344–356, Apr. 2019.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[5] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, Mar. 2017.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Thing J.*, vol. 5, no. 1, pp. 450–465, Sep. 2018.

[7] Q. Zhang and H. Li, "MOEA/D: A multi-objective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Nov. 2007.

[8] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.

[9] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symposium on Information Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 1451–1455.

[10] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Area. Comm.*, vol. 34, no. 12, pp. 3590–3605, Sep. 2016.

[11] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Nov. 2015.

[12] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Apr. 2017.

[13] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.

[14] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *Proc. IEEE 35th Int. Conf. on Computer Commun.* (*INFOCOM*), San Francisco, CA, USA, Apr. 2016, pp. 1–9.

[15] M. Masoudi, B. Khamidehi, and C. Cavdar, "Green cloud computing for multi cell networks," in *Proc. IEEE Wireless Commun. and Networking Conf.* (*WCNC*), San Francisco, CA, USA, Mar. 2017, pp. 1–6.

[16] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Mar. 2017.

[17] S. E. Mahmoodi, R.N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr. 2019.

[18] X. Xu, Y. Li, T. Huang, Y. Xue, K. Peng, L. Qi, and W. Dou, "An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks," *J. Netw. Comput. Appl.*, vol. 133, pp. 75–85, May 2019.

[19] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6398–6409, Jan. 2018.

[20] B. Huang, Z. Li, P. Tang, S. Wang, J. Zhao, H. Hu, W. Li, and V. Chang, "Security modeling and efficient computation offloading for service workflow in mobile edge computing," *Future Gener. Comp. Sy.*, vol. 97, pp. 755–774, Aug. 2019.

[21] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, Jan. 2018.

[22] 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. 8, pp. 708–719, Jul. 2018.

[23] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE ACM/Trans. Network.,* vol. 26, no. 6, pp. 2651–2664, Oct. 2018.

[24] Z. Kuang, Y. Shi, S. Guo, J. Dan, and B. Xiao, "Multi-user offloading game strategy in OFDMA mobile cloud computing system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 12, pp. 12190–12201, Dec. 2019.
[25] 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. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, Dec. 2015.
[26] J. Zhang, X. Hu, Z. Ning, E. C. H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency trade-off for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Dec. 2018.
[27] H. Peng, W. Wen, M. Tseng, and L. Li, "Joint optimization method for task scheduling time and energy consumption in mobile cloud computing environment," *Appl. Soft Comput.*, vol. 80, pp. 534–545, Jul. 2019.
[28] 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, Apr. 2019.
[29] Q. Wang, S. Guo, J. Liu, and Y. Yang, "Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing," *Sustain. Comput. Infor.*, vol. 21, pp. 154–164, Mar. 2019.
[30] L. Cui, C. Xu, S. Yang, J. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, "Joint optimization of energy consumption and latency in mobile edge computing for Internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791–4803, Sep. 2019.
[31] H. Xing, Z. Wang, T. Li, H. Li, and R. Qu, "An improved MOEA/D algorithm for multi-objective multicast routing with network coding," *Appl. Soft Comput.*, vol. 59, pp. 88–103, Oct. 2017.
[32] W. Xu, C. Chen, S. Ding, and P. M. Pardalos, "A bi-objective dynamic collaborative task assignment under uncertainty using modified MOEA/D with heuristic initialization," *Expert Syst. Appl.*, vol. 140, pp. 1–24, Feb. 2020.
[33] C. Wang, W. Zhao, W. Li, and L. Yu, "Multi-objective optimisation of electro–hydraulic braking system based on MOEA/D algorithm," *IET Intell. Transp. Sy.*, vol. 13, no. 1, pp. 183–193, Jan. 2019.
[34] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach," *IEEE Internet Things J.*, pp. 1–12, 2020, doi: 10.1109/JIOT.2019.2943373.
[35] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.
[36] Y. Liu, S. Wang, Q, Zhao, S. Du, A. Zhou, and X. Ma, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet Things J.*, pp. 1–11, 2020, doi: 10.1109/JIOT.2020.2972041.
[37] W. Aalst and K. Hee, "Workflow management: Models, methods, and systems," *The MIT Press*, 2004.
[38] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4854–4866, Jun. 2019.
[39] L. Antonio and C. C. Coello, "Coevolutionary multiobjective evolutionary algorithms: Survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 851–865, Oct. 2018.
[40] A. Trivedi, D. Srinivasan, K. Sanyal, and A. Ghosh, "A survey of multiobjective evolutionary algorithms based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 21, no. 3, pp. 440–462, Sep. 2017.
[41] B. H. Nguyen, B. Xue, P. Andreae, H. Ishibuchi, and M. Zhang, "Multiple reference points-based decomposition for multiobjective feature selection in classification: Static and dynamic mechanisms," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 170–184, Feb. 2020.
[42] G. Zhang, Y. Hu, J. Sun, and W. Zhang, "An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints," *Swarm Evol. Comput.*, vol. 54, pp. 1–15, May 2020.
[43] M. Wu, K. Li, S. Kwong, Q. Zhang, and J. Zhang, "Learning to decompose: A paradigm for decomposition-based multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 376–390, Aug. 2019.
[44] H. Li, J. Deng, Q. Zhang, and J. Sun, "Adaptive epsilon dominance in decomposition-based multiobjective evolutionary algorithm," *Swarm Evol. Comput.*, vol. 45, pp. 52–67, Mar. 2019.
[45] H. Liu, J. Pu, L. Yang, M. Lin, D. Yin, Y. Guo, and X. Chen, "A holistic optimization framework for mobile cloud task scheduling," *IEEE Trans. Sus. Comput.*, vol. 4, no. 2, pp. 217–230, Oct. 2019.
[46] W. Kang and J. Chung, "Power-and time-aware deep learning inference for mobile embedded devices," *IEEE Access*, vol. 7, pp. 3778–3789, Dec. 2019.
[47] S. Srichandan, T. Kumar, and S. Bibhudatta, "Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm," *Future Comput. Inform. J.*, vol. 3, no. 2, pp. 210–230, Dec. 2018.
[48] T. Wu, H. Gu, J. Zhou, T. Wei, X. Liu, and M. Chen, "Soft error-aware energy-efficient task scheduling for workflow applications in DVFS-enabled cloud," *J. Syst. Architect.*, vol. 84, pp. 12–27, Mar. 2018.
[49] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, "Probability and statistics for engineers and scientists," *Pearson Educ.*, 2007.
[50] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Ann. Math. Stat.*, vol. 11, no. 1, pp. 86–92, Mar. 1940.
[51] L. Wang and X. Zheng, "A knowledge-guided multi-objective fruit fly optimization algorithm for the multi-skill resource constrained project scheduling problem," *Swarm Evol. Comput.*, vol. 38, pp. 54–63, Feb. 2018.

Fuhong Song received the M.Eng. degree from Southwest Jiaotong University, Chengdu, China, in 2018. He is currently pursuing the Ph.D. degree at the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China.

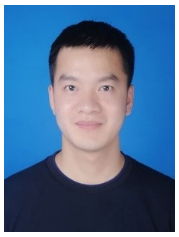
His current research interests include mobile edge computing, multiobjective optimization, and machine learning.

Huanlai Xing received the B.Eng. degree in communications engineering from Southwest Jiaotong University, Chengdu, China, in 2006, the M.Eng. degree in electromagnetic fields and wavelength technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree in computer science from University of Nottingham, Nottingham, U.K., in 2013.

He is currently an Associate Professor with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China. His current research interests include mobile edge computing, network function virtualization, software defined networks, and evolutionary computation.

Shouxi Luo received the bachelor's degree in communication engineering and the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 2011 and 2016, respectively.

He is currently a Lecturer with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China. His current research interests include data center networks, software-defined networking, and networked systems.



Dawei Zhan received the B.Eng. degree and the Ph.D. degree in School of Naval Architecture and Ocean Engineering from Huazhong University of Science and Technology, Wuhan, China, in 2012 and 2018, respectively.

His current research interests include evolutionary algorithms, surrogate modeling, and surrogate-based optimization.

Penglin Dai received the B.S. degree in mathematics and applied mathematics and the Ph.D. degree in computer science from Chongqing University, Chongqing, China, in 2012 and 2017, respectively.

He is currently an Assistant Professor with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China. His current research interests include intelligent transportation systems and vehicular cyber-physical systems.

Rong Qu received the B.Sc. in Computer Science and Its Applications from Xidian University, Xian, China in 1996 and Ph.D. in Computer Science from The University of Nottingham, Nottingham, U.K., in 2003.

She is an associated editor at IEEE Computational Intelligence Magazine, IEEE Transactions on Evolutionary Computation, Journal of Operational Research Society and PeerJ Computer Science. She is a Senior IEEE Member since 2012 and the Vice-Chair of Evolutionary Computation Task Committee at IEEE Computational Intelligence Society. Her current research interests include the modeling and automated design of optimization algorithms for combinatorial optimization problems including logistics transport scheduling, personnel scheduling, network routing, portfolio optimization and timetabling by using evolutionary algorithms, mathematical programming, constraint programming in operational research and artificial intelligence. The hybrid techniques integrated with knowledge discovery and machine learning provide intelligent decision support for real-world complex problems at SMEs, hospitals, education and industry.