



# Cost-aware workflow offloading in edge-cloud computing using a genetic algorithm

Somayeh Abdi<sup>1,2</sup> · Mohammad Ashjaei<sup>2</sup> · Saad Mubeen<sup>2</sup>

Accepted: 2 July 2024  
© The Author(s) 2024

## Abstract

The edge-cloud computing continuum effectively uses fog and cloud servers to meet the quality of service (QoS) requirements of tasks when edge devices cannot meet those requirements. This paper focuses on the workflow offloading problem in edge-cloud computing and formulates this problem as a nonlinear mathematical programming model. The objective function is to minimize the monetary cost of executing a workflow while satisfying constraints related to data dependency among tasks and QoS requirements, including security and deadlines. Additionally, it presents a genetic algorithm for the workflow offloading problem to find near-optimal solutions with the cost minimization objective. The performance of the proposed mathematical model and genetic algorithm is evaluated on several real-world workflows. Experimental results demonstrate that the proposed genetic algorithm can find admissible solutions comparable to the mathematical model and outperforms particle swarm optimization, bee life algorithm, and a hybrid heuristic-genetic algorithm in terms of workflow execution costs.

**Keywords** Workflow offloading · Edge-cloud computing · Mathematical programming · Genetic algorithm · Cost minimization · Security-aware

---

Mohammad Ashjaei and Saad Mubeen have contributed equally to this work.

---

✉ Somayeh Abdi  
somayeh.abdi@mdu.se  
Mohammad Ashjaei  
Mohammad.ashjaei@mdu.se  
Saad Mubeen  
saad.mubeen@mdu.se

<sup>1</sup> Assistant Professor, Department of Computer Engineering, Eslam Abad-E-Gharb Branch, Islamic Azad University, Eslam Abad-E-Gharb, Iran

<sup>2</sup> Department of Networked and Embedded Systems, Mälardalen University, Västerås, Sweden

## 1 Introduction

With the prosperity of the internet of things (IoT), the edge-cloud computing continuum has received considerable attention in the business and research communities as it integrates fog/cloud servers to meet the QoS requirements of the user demands, such as deadlines [1, 2]. This computing paradigm provides distributed and scalable computing resources to address some of the challenges of cloud computing, such as high latency, unpredictable bandwidth, and security issues, by locating fog servers closer to the edge devices [3]. When computing demands extend beyond an edge device, real-time tasks can be offloaded to the fog servers located closer to the edge devices which leads to reducing the latency associated with sending data [4, 5]. However, the storage and computing capacity of fog servers are limited, and compute-intensive tasks should be offloaded to the cloud to reduce the computational burden of fog servers. This computing paradigm provides additional computational power, storage capacity, scalability, and resilience which are crucial requirements for various systems such as industrial IoT [6], smart healthcare [7], smart grids [8], smart cities [9], flying vehicles such as drones [10], and autonomous vehicles [11]. In these systems, as the level of autonomy increases, more computationally intensive tasks need to be performed, and edge devices alone are insufficient to meet the requirements of the tasks.

Workflow offloading is a prominent problem in edge-cloud computing. A workflow refers to a set of tasks with data dependencies between them that perform a specific function. Workflow offloading refers to selecting appropriate computing resources from the fog or cloud layer to execute workflow tasks while meeting their QoS requirements and data dependencies [12]. QoS requirements in workflow scheduling refer to the quality of service criteria that must be met for tasks. In this paper, meeting the deadline of a workflow and the security requirements of tasks are considered as QoS requirements. Fog servers are located closer to the edge devices and users; by offloading tasks with high-security concerns to the fog server, sensitive data do not need to travel through the entire network to reach a central cloud server. It reduces the risk of unauthorized access during transmission. In some cases, compliance requirements necessitate that certain types of data remain within a specific geographic location or under strict control. Fog computing enables compliance with these regulations by keeping high-security data local, rather than transmitting it to a distant cloud server. Therefore, the unique advantages of fog computing, such as reduced transmission risks, localized control, and compliance with privacy regulations, make it an appropriate computing layer for executing tasks with high-security requirements [3, 13–15]. Workflow offloading can also consider the security requirements of tasks, allowing tasks with high-security concerns to be offloaded to the fog servers, while tasks with low-security concerns can be offloaded to the cloud.

Offloading tasks to the proper computing resources in the fog or the cloud layer have been a challenge. Such task offloading significantly impact on meeting QoS requirements of tasks and utilizing the limited capacity of fog servers. To

solve a workflow offloading problem, heuristic and meta-heuristic algorithms, mathematical models, and machine learning techniques are the most common problem-solving approaches. These approaches are different in the aspect of solution quality, time complexity, and the ability to detect an infeasible problem [16]. With advancements in parallel processing and increased computing power, the use of mathematical programming models has become more prominent in solving scientific and engineering optimization problems [17–20]. Workflow offloading is one such optimization problem and formulating it with mathematical programming models allows us to represent its essential components, including decision variables, constraints, and the objective function. Once an optimization model is formulated, it can be solved using optimization solvers like CPLEX and Gurobi, which are capable of generating globally optimal solutions for many real-world problems. Moreover, using mathematical programming also provides valuable insight into how close alternative solutions are to the optimal solution. Heuristic and meta-heuristic algorithms find good solutions in an acceptable time, but they do not guarantee the optimal solution. Furthermore, they cannot deduce whether an optimization problem is infeasible, nor do they guarantee to find a solution if one exists. Therefore, a hybrid solution of mathematical optimization and these algorithms can be used to solve a complex problem.

The complexity of workflow offloading increases when considering workflow deadlines and the security requirements of tasks, in addition to managing the data dependencies between tasks. This paper addresses this critical challenge by proposing a novel mathematical programming model and a genetic algorithm for optimizing workflow offloading in edge-cloud computing. It formulates the problem of workflow offloading in edge-cloud computing with a nonlinear programming model. The objective function is to minimize the monetary cost of executing a workflow while satisfying constraints related to the data dependency among the tasks, the workflow deadline, and the security requirements of tasks. This model provides precise definitions of the problem's components, including decision variables, constraints, and objective functions. This clarity helps in understanding the problem thoroughly which is essential for analyzing and optimizing workflow offloading. It also helps identify and understand the relationships between different decision variables and constraints. Moreover, inspired by the proposed mathematical model and heuristic algorithms, we propose a genetic algorithm for the workflow offloading problem that efficiently finds feasible solutions. We define problem-specific crossover and mutation operators to generate solutions that satisfy the data dependencies between tasks, the workflow deadlines, and the security requirements of the tasks. Furthermore, we integrate the stochastic nature of generating the initial population with heuristic rules to generate feasible solutions and accelerate the convergence of the algorithm.

Genetic algorithms (GAs) are well-suited for complex optimization problems with large search spaces and multiple constraints, such as task scheduling. They are an effective approach for solving complex optimization problems, as they can explore large search spaces due to their stochastic nature and crossover/mutation operations. This ability makes them a good fit for our workflow scheduling problem. Moreover, GAs are known for their robustness and flexibility in handling various types of optimization problems, including those with nonlinear, and discrete

characteristics [21]. Our workflow scheduling problem involves such complexities, making GA an appropriate choice. With well-understood operators (selection, crossover, mutation), GAs are highly flexible and can be adapted to different scheduling scenarios. The main contributions of this paper are as follows:

- We propose a nonlinear mathematical model that minimizes the monetary cost of executing a workflow including the cost of renting computing nodes and data transfer costs. The model also satisfies the data dependencies between tasks, the workflow deadline, and the security requirements of tasks.
- We develop a genetic algorithm with specialized mutation and crossover operations tailored to satisfy the constraints of data dependency, workflow deadline, and security requirements.
- We perform a set of experiments based on existing workflows to compare the performance of the genetic algorithm with the proposed mathematical model and some meta-heuristic algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related works in the area of task offloading in edge-cloud computing. Section 3 introduces the system model and workflow model. Section 4 presents the proposed mathematical model for the workflow offloading problem, including the objective function and constraints of the model. Section 5 presents the proposed genetic algorithm. The simulation results of the proposed approaches are reported in Sect. 6, and finally, Sect. 7 concludes the paper.

## 2 Related work

Task offloading in edge-cloud computing has been studied with different improvement criteria such as energy consumption, completion time, and monetary cost [22]. Moreover, various optimization techniques such as mathematical programming, heuristic and meta-heuristic algorithms, machine learning algorithms, and queuing theory have been used to solve this problem [23, 24].

Some research has focused on the offloading of tasks, where the tasks are independent of each other and there are no data dependencies among the tasks. The work in [25] proposes a particle swarm optimization algorithm that minimizes the cost and makespan of executing tasks in fog-cloud computing. This paper assumes that applications have the Bag-of-Tasks (BoT) structure, where independent tasks are submitted to the fog broker, which communicates directly with the mobile users. The broker decides to execute tasks on the fog or cloud nodes based on the availability of fog resources. The work in [26] proposes a genetic algorithm for task offloading in edge-cloud computing with the objective of minimizing energy consumption. This paper considers the energy consumption for processing tasks and the energy consumption for data transmission. It is assumed that the tasks are parallel and there are no data dependencies among tasks. The work in [27] proposes a bee life algorithm (BLA) for parallel task scheduling in fog computing. It minimizes the cost of processing tasks while considering the memory requirements of tasks. This

paper assumes that several tasks can be assigned to the same fog node and the proposed algorithm identifies the order of executing tasks on the assigned nodes. The proposed BLA algorithm starts with generating an initial population randomly and uses crossover and mutation operators to generate offspring solutions. The work in [28] formulates the problem of Bag-of-Tasks (BoT) scheduling as a permutation-based optimization problem and proposes a scheduling algorithm that minimizes the monetary cost under a deadline constraint. It uses a modified version of the genetic algorithm to generate different permutations for tasks at each scheduling round. Then, the tasks are assigned to VMs that have sufficient resources and achieve the minimum execution cost. The work in [29] proposes a task scheduling method based on reinforcement learning, a Q-learning algorithm, to minimize the completion time of parallel tasks. It considers the delay of transmission, execution, and queuing time to formulate the completion time of tasks. It also applies digital twins to simulate the results of different actions made by an agent, so that one agent can try multiple actions at a time, or, similarly, multiple agents can interact with the environment in parallel.

Some research has focused on offloading dependent tasks to fog or cloud servers. The work in [30] formulates the workflow scheduling in edge-cloud computing with the objective of makespan optimization with mathematical programming. It also presents a heuristic algorithm that finds near-optimal solutions for the scheduling problem. The mentioned heuristic algorithm prioritizes tasks based on their workload and data dependency among tasks then selects a proper edge or cloud server based on the earliest finish time policy. The work in [31] proposes multi-objective workflow offloading in Multi-access Edge Computing (MEC) based on a Meta-Reinforcement Learning (MRL) algorithm. The objective is to minimize task latency, resource utilization rate, and energy consumption when a local scheduler decides to offload the tasks to the MEC host. The work in [32] proposes a genetic algorithm that minimizes the makespan of dependent tasks. The work in [33] proposes a genetic algorithm that minimizes the energy consumption and makespan of the workflows that are generated by IoT devices. Nonetheless, these works do not consider the complexity emerging from the heterogeneity of fog and cloud resources and workflow requirements such as required memory, or security constraints.

Cost modeling is another critical improvement criterion in edge-cloud computing [37], and some related works focus on task offloading with cost minimization objectives. The work in [34] proposes a mathematical model for task offloading in edge-cloud computing with the objective of energy and computational cost optimization. It only considers the computational cost of tasks, and the data transfer cost is not considered. The mentioned paper also proposes a branch-and-bound method for solving the proposed mathematical model. The work in [35] proposes a framework for resource provisioning in cloud computing with the cost minimization objective under deadline constraints. It considers data dependency between tasks and proposes a particle swarm-based algorithm to find the optimal assignment of tasks to the virtual machines in cloud computing. The work in [36] utilizes queuing theory to study energy consumption and cost when computations are offloaded to fog or cloud computing. The mentioned work considers different queuing models in edge-cloud computing. It considers the  $M/M/1$  queue at the mobile device layer, the  $M/M/c$

**Table 1** Summary of related works about task offloading in Edge-cloud computing

Article	Main idea	Task requirements	Improved criteria
Nguyen et al. [25]	Particle swarm optimization	–	Cost Makespan
Nan et al. [26]	Genetic algorithm	–	Energy consumption
Bitam et al. [27]	Bee life	Memory	Cost
Abohamama et al. [28]	Heuristic GA-based algorithm	Deadline	Execution cost
Wang et al. [29]	Reinforcement learning Digital twin	–	Completion time of tasks
Lou et al. [30]	List-based heuristic algorithm	Data dependency	Makespan
Liu et al. [31]	Reinforcement Learning	Data dependency	Energy consumption Task latency
Xu et al. [32]	Genetic algorithm	Data dependency	Makespan
Saeed et al. [33]	Genetic algorithm	Data dependency	Makespan Energy consumption
Haber et al. [34]	Mathematical model Branch and bound method	–	Computational cost Energy consumption
Alsurdeh et al. [35]	Particle swarm optimization	Data dependency Deadline	Cost
Liu et al. [36]	Queuing theory	–	Energy consumption Cost

queue at the fog node with a defined maximum request rate, and the  $M/M/\infty$  queue at the cloud layer. Based on the applied queue theory, it formulates a multi-objective optimization problem for minimizing energy consumption, delay, and monetary cost of parallel tasks. Table 1 summarizes the main idea, the task requirements, and improvement criteria that are considered in task offloading problems in related works.

Meeting the deadline of a workflow while satisfying the data dependency between tasks has been a challenge in edge-cloud computing. In addition, other workflow requirements such as security concerns and memory requirements are important constraints to consider. To address this problem, this paper proposes a nonlinear mathematical programming model. The objective is to minimize the cost while satisfying the above constraints. Moreover, the paper proposes a genetic algorithm to find a near-optimal solution while satisfying the mentioned constraints.

### 3 System model

The proposed edge-cloud computing continuum is depicted in Fig. 1, which comprises an edge layer, a fog layer, and a cloud layer. The first layer includes heterogeneous IoT devices such as smartphones and healthcare devices, and a smart gateway (or access point) which is located at the edge of the network. Since an edge device has limited computing and storage capacity, it can request some demands

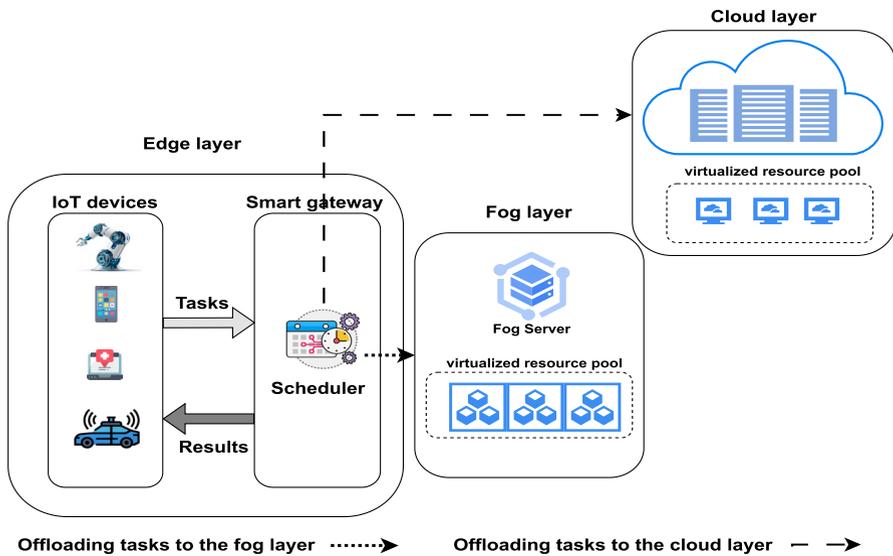


Fig. 1 Overview of the system model

formed as dependent tasks, called a workflow, to run on fog or cloud nodes. These demands are received by the smart gateway. The main part of the proposed model is an entity called a scheduler, which is deployed in the smart gateway. The optimization algorithm is executed in the scheduler, and it decides on the computing layer where the tasks of a workflow will be executed. These decisions are based on the workflow requirements and the available resource capacity of edge, fog, and cloud layers. The fog layer provides limited computing resources closer to the source of generated workflows. The cloud layer is the topmost layer, and it is a remote centralized cloud that provides scalable computing and storage resources. Although communication latency is higher in the latter, in order to meet the workflow deadline, computationally intensive tasks should be submitted to the cloud layer, as it provides higher computing capacities than the fog layer. In the following subsections, the resource model and workflow model are explained. Table 2 shows the main notations used in this paper.

### 3.1 Resource model

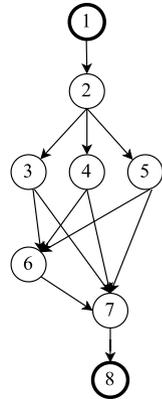
As mentioned, we consider a computing continuum comprising an edge layer, a fog layer, and a cloud layer, depicted in Fig. 1. These computing layers provide computing resources; either containers or virtual machines. For the sake of simplicity, we call a computing resource a node, and an edge device provides only one node. Notations of computing layers and provided nodes are described in Table 2. Generally, it is assumed that three computing layers participate in the edge-cloud computing continuum, where the notation  $CL = \{CL_{edge} \cup CL_{fog} \cup CL_{cloud}\}$  is the set of computing

**Table 2** Model parameters and their description

Notation	Description
Notations of a workflow	
$W = (T, D)$	Workflow $W$ indicating by task set $T$ and dependency matrix $D$ .
$T = \{\tau_1, \tau_2, \dots, \tau_n\}$	A set of $n$ tasks in the workflow $W$
$D \in \mathbb{R}_{ T  \times  T }$	Dependency matrix $D$ ; element $d_{ij}$ in $D$ is the amount of data that must be transferred from task $\tau_i$ to $\tau_j$
$et_i$	The computation size of task $\tau_i$ , i.e., Million Instructions (MI)
$mem_i$	The minimum required memory (RAM) for executing task $\tau_i$
$ts_i$	Security tag of task $\tau_i$
$r_i$	Rank of task $\tau_i$
$pre_i$	Set of predecessor task(s) of task $\tau_i$
$suc_i$	Set of successor task(s) of task $\tau_i$
$ids_1$	Input data size of task $\tau_1$ in data unit size
$ods_n$	Output data size of task $\tau_n$ in data unit size
$D^w$	Deadline of the workflow $W$
Notation of computing layers and computing nodes	
$CL = \{CL_{edge} \cup CL_{fog} \cup CL_{cloud}\}$	A set of computing layers participating in the edge-cloud continuum
$CL_k$	Computing layer $k$
$rs_k$	Security tag of $CL_k$
$IT_k \implies \{I_{k1}, I_{k2}, \dots, I_{kq}\}$	The set of instance types provided by $CL_k$
$I_{kj}$	$j$ th instance type provided by $CL_k$
$NP_{kj} \implies \{N_{kj1}, N_{kj2}, \dots, N_{kj p}\}$	A node pool provided for instance type $I_{kj}$
$N_{kjl}$	$l$ th node of instance type $I_{kj}$
$pr_{kj}$	The cost of renting a node of instance type $I_{kj}$ in the time unit $\Delta_r$ .
$pf_{kj}$	The CPU processing capacity of a node of $I_{kj}$ , i.e., million instruction per second (MIPS)
$bw_{kj}$	Communication bandwidth of a node of instance type $I_{kj}$
$ram_{kj}$	The memory capacity (RAM) of a node of instance type $I_{kj}$
$bw_{(k,k')}$	The communication bandwidth between computing layers $k$ and $k'$
$dtc$	The price of data transfer cost between two different computing layers in the data unit

layers. The notation  $CL_k$  indicates computing layer  $k$ . Since we propose a security-aware task offloading model, we consider a security tag for each computing layer and task. The notation  $rs_k \in \{s_{pu}, s_{sp}, s_{pr}\}$  indicates the security tag of  $k$ th computing layer, i.e.,  $CL_k$ . Here, the cloud layer has a public security tag, i.e.,  $rs_{cloud} = s_{pu}$ , the fog layer has a semi-private security tag, i.e.,  $rs_{fog} = s_{sp}$ , and the edge layer has a private security tag, i.e.,  $rs_{edge} = s_{pr}$ , where  $s_{pr} < s_{sp} < s_{pu}$ , and a smaller security tag indicates that the task or computing layer has a higher security concern. In this work, we consider  $s_{pu} = 3$ ,  $s_{sp} = 2$ , and  $s_{pr} = 1$ , where  $s_{pr} < s_{sp} < s_{pu}$ . In the workflow model, Sect. 3.2, the security constraint for assigning tasks in the workflow to the computing layers is explained in detail.

**Fig. 2** A sample DAG representing a workflow



Moreover,  $CL_k$  provides a set of instance types with different prices and performance levels, i.e.,  $IT_k \implies \{I_{k1}, I_{k2}, \dots, I_{kq}\}$ . The notation  $I_{kj}$  indicates  $j$ th instance type in the  $k$ th computing layer. For each instance type  $I_{kj}$ , a pool of  $p$  nodes is provided, i.e.,  $NP_{kj} \implies \{N_{kj1}, N_{kj2}, \dots, N_{kjp}\}$ . The notation  $N_{kjl}$  indicates  $l$ th node of instance type  $I_{kj}$ . Nodes of the same type are provisioned with the same characteristics. The notations used to describe the characteristics of instance type  $I_{kj}$  are as follows:

$$I_{kj} : \langle pr_{kj}, ram_{kj}, bw_{kj}, pf_{kj} \rangle$$

The notation  $pr_{kj}$  denotes the price of renting a node of instance type  $I_{kj}$  per time unit  $\Delta_t$ . The notation  $ram_{kj}$  denotes the memory capacity (RAM) of a node of instance type  $I_{kj}$ . The communication bandwidth of a node of instance type  $I_{kj}$  is denoted by  $bw_{kj}$ . The notation  $pf_{kj}$  indicates the CPU processing capacity of a node of instance type  $I_{kj}$ , i.e., the number of instructions that each node of type  $I_{kj}$  can process per second (MIPS). Moreover, the notation  $bw_{(k,k')}$  indicates the communication bandwidth between computing layers  $k, k'$ . It is assumed that the data transfer between nodes in the same computing layer is free, and only data transfer between different computing layers is charged. The notation  $dtc$  indicates the cost of transferring data per data unit, between two different computing layers.

### 3.2 Workflow model

It is assumed that an IoT device generates applications that are composed of dependent tasks [38]. Different IoT applications such as video analysis [39], object detection [40], navigator [41], and cognitive assistance [42] are composed of dependent tasks. These applications are called workflows and their structure can be modeled as a DAG (Direct Acyclic Graph), depicted in Fig. 2. A DAG is a way of representing data dependencies among tasks that must be executed in a particular order, where the input data of a task is the output data of one or more previous tasks.

In this paper, workflow  $W = (T, D)$  is represented by  $T$  and  $D$ , where  $T$  is a set of  $n$  tasks, i.e.,  $T = \{\tau_1, \tau_2, \dots, \tau_n\}$ , and  $D \in \mathbb{R}_{|T| \times |T|}$  is a dependency matrix that indicates the data dependencies among tasks.  $D$  is an upper triangular matrix and all elements in the diagonal are zero. Element  $d_{ij}$  ( $i < j$ ) represents data dependency among tasks  $\tau_i, \tau_j$ . If  $d_{ij} = 0$ , there is no data dependency between  $\tau_i, \tau_j$ . Otherwise,  $d_{ij}$  indicates the amount of data that must be transferred from task  $\tau_i$  to  $\tau_j$ , and task  $\tau_j$  can start after task  $\tau_i$  transmits its output data to it. Here,  $\tau_i$  and  $\tau_j$  represent predecessor and successor tasks, respectively. In this work, it is assumed that workflow  $W$  has a deadline  $\mathcal{D}^w$ , and all the tasks in the workflow should meet the deadline. The notations used to describe the characteristics of task  $\tau_i$  are as follows:

$$\tau_i : \langle et_i, mem_i, ts_i, r_i, pre_i, suc_i \rangle$$

The notation  $et_i$  is the computation size of task  $\tau_i$ , i.e., million instructions (MI). The minimum required memory (RAM) for executing task  $\tau_i$  is denoted by  $mem_i$ . The notation  $ts_i \in \{s_{pu}, s_{sp}, s_{pr}\}$  is the security tag of task  $\tau_i$ . We consider three security tags for tasks including public security tag, i.e.,  $s_{pu}$ , semi-private security tag, i.e.,  $s_{sp}$ , and private security tag, i.e.,  $s_{pr}$ , and  $s_{pr} < s_{sp} < s_{pu}$ , where a smaller security tag indicates that the task has a higher security concern. Therefore, each task can have one of the following values: If  $ts_i = s_{pr}$ , task  $\tau_i$  is private, and it can be executed only on the edge layer. If  $ts_i = s_{sp}$ , task  $\tau_i$  is semi-private, and it can only be executed on the nodes in the edge or fog layer. If  $ts_i = s_{pu}$ , task  $\tau_i$  is public, and it can be executed on the nodes in the edge, fog, or cloud layer. As mentioned in Sect. 3.1, we consider  $s_{pu} = 3, s_{sp} = 2, s_{pr} = 1$ , where  $s_{pr} < s_{sp} < s_{pu}$ . Accordingly, security constraint for assigning task  $\tau_i$  to computing layer  $k$  can be represented as follows:

$$SC_{i,k} = \begin{cases} k \leftarrow \{CL_{edge}\} & ts_i = 1 \\ k \leftarrow \{CL_{edge}, CL_{fog}\} & ts_i = 2 \\ k \leftarrow \{CL_{edge}, CL_{fog}, CL_{cloud}\} & ts_i = 3 \end{cases} \quad (1)$$

According to this constraint, a private task can only be executed in the edge layer while a public task can be executed in the edge, fog, or cloud layer. In addition, we assume that each task in the workflow has a rank. The notation  $r_i$  indicates the rank of task  $\tau_i$ . The rank of a task is determined based on its index, i.e.,  $r_i = i$ . In our proposed model, if two independent tasks are assigned to the same node, the task with a lower rank will be executed first.

The notation  $pre_i$  is the set of predecessor tasks of  $\tau_i$ , i.e.,  $pre_i = \{\tau_t \mid d_{it} \neq 0\}$ . The notation  $suc_i$  is the set of successor tasks of  $\tau_i$ , i.e.,  $suc_i = \{\tau_t \mid d_{it} \neq 0\}$ . Task  $\tau_1$  is the entry task of the workflow and has no predecessor task(s). Task  $\tau_n$  is the exit task of the workflow and has no successor task(s). The notations  $ids_1$  and  $ods_n$  indicate the input and output data size of tasks  $\tau_1$  and  $\tau_n$ , respectively. Figure 2 shows an example of a workflow with 8 tasks, where  $\tau_1$  is the entry task, and  $\tau_8$  is the exit task. The successors of  $\tau_1$ , i.e.,  $suc_1 = \{\tau_2\}$ , while the predecessor of  $pre_6$ , i.e.,  $pre_6 = \{\tau_3, \tau_4, \tau_5\}$ .

## 4 Problem formulation

In this paper, we formulate the problem of workflow offloading in an edge-cloud computing continuum as a nonlinear mathematical programming model (MPM) and propose a genetic algorithm that finds feasible solutions for the scheduling problem. In this section, the constraints of the task offloading problem and the decision variables of the mathematical model are introduced. Then, the objective function of the model is described. The proposed model fulfills constraints related to data dependencies among tasks, the security requirements of tasks, the memory required to execute tasks, and the deadline of the workflow. The objective is to minimize the monetary cost of executing the workflow, which consists of the cost of renting nodes to execute the tasks in the workflow and the cost of transferring data between dependent tasks.

It is assumed that workflow  $W$  is submitted to the scheduler. As shown in Fig. 1, the optimization algorithm is run in the scheduler, and it selects the appropriate computing layers and nodes from the edge-cloud computing continuum to execute the tasks in the workflow.

### 4.1 Assignment constraint

Workflow  $W$  contains  $n$  tasks, and the assignment of a task refers to allocating the task to a computing node in an edge-fog-cloud environment. Therefore, a schedule  $\Phi$  is a set of assignments, i.e.,  $\Phi = \{A_i \mid i = 1, \dots, n\}$  and the assignment of task  $\tau_i$  is represented as follows:

$$A_i : < \tau_i, N_{kjl}, ST_i, RT_i >$$

This assignment indicates that task  $\tau_i$  is assigned to  $l$ th node of instance type  $I_{kj}$ , i.e.,  $N_{kjl}$  and  $ST_i$  and  $RT_i$  are the start time and run time of the task on the assigned node, respectively. To formulate assignment  $A_i$ , we define decision variable  $y_{ikjl}$  where  $y_{ikjl} = 1$  if and only if task  $\tau_i$  is assigned to node  $N_{kjl}$ ; otherwise, it attains 0. The assignment constraint is formulated in Eq. (2). This constraint guarantees that all tasks in the workflow are scheduled, and each task is to be scheduled only once.

$$\sum_{k \in CL} \sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} = 1 \quad \forall i \in T \tag{2}$$

A schedule  $\Phi$  is considered feasible if and only if the following constraints are satisfied.

## 4.2 Memory constraint

A task can be assigned to an instance type that satisfies the minimum required memory for its execution. This constraint is formulated in Eq. (3). In this equation,  $\sum_{l \in NP_{kj}} y_{ikjl} = 1$  if and only if task  $\tau_i$  is assigned to a node of instance type  $I_{kj}$ .

$$\sum_{l \in NP_{kj}} y_{ikjl} \cdot mem_i \leq ram_{kj} \quad \forall i \in T, \forall k \in CL, \forall j \in IT_k \quad (3)$$

## 4.3 Security constraint

Since we propose a security-aware task scheduling model in the edge-cloud computing continuum, security tags are considered for tasks and computing layers in Sects. 3.1 and 3.2. Equation (4) ensures that each task is assigned to a computing layer that satisfies its security requirements according to Eq. (1).

$$\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} \cdot rs_k \leq ts_i \quad \forall i \in T, \forall k \in CL \quad (4)$$

In this equation,  $\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} = 1$  if and only if task  $\tau_i$  is assigned to a node in  $k$ th computing layer. According to this constraint, a private task can only be executed in the edge layer, a semi-private task can be executed in the fog or edge layer, and a public task can be executed in the edge, fog, or cloud layer. If task  $\tau_i$  is private then  $ts_i = 1$ , and according to Eq. (4), inequality  $\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} \cdot rs_k \leq 1$  should be satisfied. Considering  $s_{pr} = 1, s_{sp} = 2, s_{pu} = 3$ , expression  $\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl}$  can only take value 1, where  $rs_k = 1$ . Since the edge layer has a private security tag, i.e.,  $rs_{edge} = 1$ , task  $\tau_i$  can only be assigned to the edge layer. If task  $\tau_i$  is public then  $ts_i = 3$ , and inequality  $\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} \cdot rs_k \leq 3$  should be satisfied. In this case, task  $\tau_i$  can be assigned to the edge, fog, or cloud layer.

## 4.4 Non-preemption constraint

In this work, we consider non-preemption resource allocation for tasks, once a task starts on a node, it will finish without any interruption, i.e., it cannot be paused or migrated until it completes. Given the assignment  $A_i$ , this constraint can be represented as:

$$M_i = ST_i + RT_i \quad \forall i \in T \quad (5)$$

In this equation, decision variable  $M_i$  indicates the completion time of task  $\tau_i$  on the allocated node in assignment  $A_i$ . According to this assignment, the run time of task  $\tau_i$  on a node of instance type  $I_{kj}$  is as follows:

$$RT_i \geq \sum_{l \in NP_{kj}} y_{ikjl} \cdot \frac{et_i}{pf_{kj}} \quad \forall i \in T, \forall k \in CL, \forall j \in IT_k \tag{6}$$

where  $et_i$  indicates the computation size of task  $\tau_i$ , and  $pf_{kj}$  represents the CPU processing capacity of a node of instance type  $I_{kj}$  which task  $\tau_i$  is assigned to it. In this equation,  $\sum_{l \in NP_{kj}} y_{ikjl} = 1$  if and only if task  $\tau_i$  is assigned to a node of instance type  $I_{kj}$ .

#### 4.5 Data dependency among tasks and precedence constraint

As mentioned in Sect. 3.2, we consider a workflow consisting of  $n$  tasks. There are data dependencies among tasks, and the notation  $pre_i$  indicates the predecessors of task  $\tau_i$ . To fulfill data dependency among tasks in a workflow, each task starts only when all its predecessor tasks have been completed and their output data have been sent to it. Here, we define the decision variable  $ReadyT_i$ , which indicates the time at which task  $\tau_i$  has received all of the input data from its predecessor(s) and is ready to start. Since the entry task in the workflow, i.e.,  $\tau_1$ , has no predecessors, we do not consider this constraint for task  $\tau_1$ . After a workflow  $W$  is released, the entry task is ready to run on a computing resource at the time of zero, thus  $ReadyT_1 = 0$ . For other tasks in the workflow, the time that a task can start must be later than the time that it receives all input data from its predecessor tasks. The data dependency among tasks can be formulated as follows:

$$ReadyT_1 = 0 \tag{7}$$

$$ReadyT_i \geq M_i + \beta_{t,i} \quad \forall i \in T - \{\tau_1\}, \quad \forall t \in pre_i \tag{8}$$

In Eq. (8), decision variable  $M_i$  indicates the completion time of task  $\tau_i$ , and  $\beta_{t,i}$  indicates the data transfer time from predecessor task  $\tau_t$  to  $\tau_i$ ,  $\forall \tau_t \in pre_i$ . Although the use of computing nodes in different layers presents some challenges, such as communication congestion, we do not model the effects of communication congestion for utilizing bandwidth between different computing layers [43]. The notation  $bw_{(k,k')}$  indicates the available bandwidth between computing layers  $k$  and  $k'$ , and we considered this bandwidth in formulating the data transfer time between dependent tasks in different computing layers. To formulate data transfer time between task  $\tau_i$  and its predecessor task  $\tau_t$ , we consider three possible cases and define decision variables  $v_{it}$ ,  $cl_{it}$  and  $\theta_{it}$ .

- In the case that task  $\tau_i$  and its predecessor task  $\tau_t$  run in the same node then the data transfer time from task  $\tau_t$  to  $\tau_i$  is 0, i.e.,  $\beta_{t,i} = 0$ . Decision variable  $v_{it}$  indicates whether tasks  $\tau_i$  and  $\tau_t$  are assigned to the same node or not. If tasks  $\tau_i$  and  $\tau_t$  are assigned to the same node then  $v_{it} = 1$ ; otherwise,  $v_{it} = 0$ . Equation (9) calculates the value of this decision variable.

$$v_{it} = y_{ikjl} \cdot y_{tkjl} \quad (9)$$

$$\forall i \in T - \{\tau_1\}, \forall t \in pre_i, \forall k \in CL, \forall j \in IT_k, \forall l \in NP_{kj}$$

- In the case that tasks  $\tau_i$  and  $\tau_t$  run in the same computing layer but different nodes, data transfer time is computed by considering the communication bandwidth of instance type  $I_{kj}$  that task  $\tau_t$  is assigned to it, and  $\beta_{t,i} = d_{it}/bw_{kj}$ . Here,  $d_{it}$  is the output data size of task  $\tau_t$  which must be transferred to  $\tau_i$ . Decision variable  $cl_{it}$  indicates whether  $\tau_i$  and  $\tau_t$  are assigned to different nodes in the same computing layer or not. If tasks  $\tau_i$  and  $\tau_t$  are assigned to different nodes in the same computing layer then  $cl_{it} = 1$ ; otherwise,  $cl_{it} = 0$ . Equation (10) calculates the value of this decision variable.

$$cl_{it} = (1 - v_{it}) \cdot \sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{ikjl} \cdot \sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{tkjl} \quad (10)$$

$$\forall i \in T - \{\tau_1\}, \forall t \in pre_i, \forall k \in CL$$

- In the case that tasks  $\tau_i$  and  $\tau_t$  run in different computing layers,  $\beta_{t,i} = d_{it}/bw_{(k,k')}$ , where  $bw_{(k,k')}$  denotes communication bandwidth between the computing layers  $k$  and  $k'$ . Decision variable  $\theta_{it}$  indicates whether tasks  $\tau_i, \tau_t$  are assigned to different computing layers or not. If tasks  $\tau_i, \tau_t$  are assigned to different computing layers then  $\theta_{it} = 1$ ; otherwise,  $\theta_{it} = 0$ . Equation (11) calculates the value of this decision variable.

$$\theta_{it} = (1 - cl_{it}) \cdot (1 - v_{it}) \quad \forall i \in T - \{\tau_1\}, \forall t \in pre_i \quad (11)$$

According to the above, the data transfer time between task  $\tau_i$  and its predecessor task  $\tau_t$  is formulated as follows:

$$\beta_{t,i} = cl_{it} \cdot d_{it}/bw_{kj} + \theta_{it} \cdot d_{it}/bw_{(k,k')} \quad \forall i \in T - \{\tau_1\}, \forall t \in pre_i \quad (12)$$

If task  $\tau_i$  and  $\tau_t$  run on the same node then  $v_{it} = 1$ ,  $cl_{it} = 0$ , and  $\theta_{it} = 0$ , so  $\beta_{t,i} = 0$ ; In this case, the data transfer time between the dependent tasks is 0. If task  $\tau_i$  and  $\tau_t$  run on different nodes in the same computing layer then  $v_{it} = 0$ ,  $cl_{it} = 1$ ,  $\theta_{it} = 0$ , so  $\beta_{t,i} = d_{it}/bw_{kj}$ . In this case, the data transfer time between dependent tasks is computed by considering the communication bandwidth of instance type  $I_{kj}$  that task  $\tau_t$  is assigned to it. If task  $\tau_i$  and  $\tau_t$  run in different computing layers then  $v_{it} = 0$ ,  $cl_{it} = 0$ ,  $\theta_{it} = 1$ , so  $\beta_{t,i} = d_{it}/bw_{(k,k')}$ . In this case, the data transfer time between dependent tasks is computed by considering the communication bandwidth between the computing layers  $k$  and  $k'$ .

#### 4.6 Assignment dependency constraint

The execution times of tasks assigned to the same node cannot overlap. This is because each node can only execute its assigned tasks sequentially. If there is data dependency among two tasks assigned to the same node, based on the data dependency formulated in Eq. (8), the tasks do not overlap. For two tasks  $\tau_i$  and  $\tau_t$

that there are no data dependencies between them and assigned to the same node, we define the assignment dependency as follows:

$$ad_i = \{\tau_t \mid y_{ikjl} \cdot y_{tkjl} = 1 \ \& \ r_t < r_i \ \& \ t \notin pre_i \ \& \ i \notin pre_t\} \tag{13}$$

where  $r_i$  indicates the rank of task  $\tau_i$ . Thus,  $ad_i$  is the set of predecessor tasks of task  $\tau_i$  that must be scheduled before it due to the assignment dependency.

We considered the data dependency and the assignment dependency in formulating the start time of tasks. The decision variable  $ST_i$  indicates the start time of task  $\tau_i$  on the assigned node. Task  $\tau_i$  starts only when it has received all data from its predecessor tasks due to data dependency and when all its predecessor tasks due to assignment dependency have been completed. Therefore, the start time of task  $\tau_i$  is formulated as follows:

$$ST_i \geq \max \{ReadyT_i, \max (M_t)\} \ \forall i \in T, \forall t \in ad_i \tag{14}$$

where  $ReadyT_i$  is the time at which task  $\tau_i$  receives all input data from its predecessor tasks due to data dependency. As mentioned,  $ad_i$  is the set of predecessor tasks of task  $\tau_i$  that must be scheduled before it due to the assignment dependency.  $M_t$  is the completion time of task  $\tau_t$  and the expression  $\max(M_t) \ \forall t \in ad_i$  is the completion time of all predecessor tasks of task  $\tau_i$  due to assignment dependency.

### 4.7 Deadline constraint

The exit task of the workflow, i.e.,  $\tau_n$ , receives the final execution results. Therefore, the completion time of a workflow is equal to the exit task’s completion time, i.e.,  $M_n$ . If the exit task is assigned to the fog or cloud layer then the data transfer time of its output data to the edge node should be considered. To satisfy the deadline constraint, the completion time of the workflow must be less or equal to the deadline of the workflow. This constraint is represented as follows:

$$M_n + \sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{nkjl} \cdot ods_n / bw_{(k,k')} \leq \mathcal{D}^w \ \forall k \in \{CL_{fog}, CL_{cloud}\} \tag{15}$$

where  $k' = \{CL_{edge}\}$  and  $\sum_{j \in IT_k} \sum_{l \in NP_{kj}} y_{nkjl} = 1$  if and only if task  $\tau_n$  is assigned to a node in the fog or cloud layer. In this case, the expression  $ods_n / bw_{(k,k')}$  indicates the data transfer time to send the output data of the last task in the workflow to the edge device. If the last task of the workflow runs in the edge layer, then, the data transfer time of the last task attains 0.

### 4.8 Objective function

A feasible schedule  $\Phi$  is evaluated by the monetary cost of executing a workflow, formulated in Eq. (16). This cost includes the cost for renting nodes, formulated in Eq. (17), and the data transfer cost, formulated in Eq. (18).

$$\begin{aligned}
& \text{Cost} = \text{Minimize } (C_c + C_d) \\
& \text{Subject to:} \\
& \text{Constraints (2)-(17), (18)}
\end{aligned} \tag{16}$$

We consider the cost of renting nodes based on the ‘pay-as-you-go’ model. According to this pricing model, the cost of renting a node is charged based on the number of time intervals ( $\Delta_t$ ) that it has been allocated to the tasks. To calculate this cost, the start and the end timestamps of renting nodes should be determined. Assuming that node  $N_{kjl}$  is allocated to the tasks, in Eq. (17),  $\max\{M_i \cdot y_{ikjl}\}$  indicates the completion time of the latest task assigned to the node, and  $\min\{ST_i \cdot y_{ikjl}\}$  indicates the start time of the earliest tasks assigned to the node. Therefore, in Eq. 17, the expression  $(\max\{M_i \cdot y_{ikjl}\} - \min\{ST_i \cdot y_{ikjl}\})$  indicates the total time of running node  $N_{kjl}$  to complete assigned tasks to it. Finally, this time is divided by the time interval  $\Delta_t$  and rounded up to determine the number of time intervals the node will rent.

$$C_c = \sum_{i \in T} \sum_{k \in CL} \sum_{j \in IT_k} \sum_{l \in NP_{kj}} pr_{kj} \cdot \left\lceil \frac{\max\{M_i \cdot y_{ikjl}\} - \min\{ST_i \cdot y_{ikjl}\}}{\Delta_t} \right\rceil \tag{17}$$

Different cloud providers consider various time intervals for charging a VM, for instance, AWS charges for EC2 instances on a per-hour basis, while there are some cloud providers such as Microsoft Azure that charge for VMs on a per-minute basis. As mentioned in Sect. 3, we consider the CPU capacity of nodes as MIPS and the task computation size as MI. In the experiments, we consider Amazon EC2 instances which are charged on a per-hour basis. Therefore, to bridge the gap between the task computation time (e.g., second) and the time interval of renting a node (e.g., hour), in Eq. (17), we incorporate the following points:

- We convert the time interval of renting a node from seconds to hours by dividing the total time of running a node by  $\Delta_t = 3600$  (the number of seconds in an hour) to get the time in hours.
- Cloud providers typically round up to the nearest hour for billing purposes. Thus, after converting the time of allocating a node to tasks to hours, we apply the ceil function to ensure that we account for the full hourly billing cycle.

The data transfer cost is formulated in Eq. (18). This cost depends on the amount of data transferred between dependent tasks. It is assumed that only the data transfer between different computing layers is charged. If dependent tasks are assigned to the same computing layer, the data transfer is free; otherwise, it is charged.

$$C_d = \sum_{i \in T - \{t1\}} \sum_{t \in pre_i} \theta_{it} \cdot d_{it} \cdot dtc \tag{18}$$

If task  $\tau_i$  and its predecessor task  $\tau_t$  are assigned to different computing layers, then,  $\theta_{it} = 1$  and the data transfer is charged; otherwise,  $\theta_{it} = 0$  and the data transfer is

free. Here,  $d_{ii}$  is the amount of data that must be transferred from task  $\tau_i$  to  $\tau_i$ , and  $d_{tc}$  is the transmission cost per data unit.

## 5 Proposed genetic algorithm

Task scheduling problems are known as NP-hard problems [44–46]. This means that there are no polynomial-time algorithms that can find optimum solutions to these problems. Therefore, this paper proposes a genetic algorithm for solving the workflow offloading problems in the edge-cloud computing continuum, called WSEFC-GA. The stochastic nature of GAs, combined with their population-based search, enhances their ability to perform a global search. This makes them particularly useful for complex problems while other algorithms such as the beluga whale optimization algorithm might get stuck in local optima [47, 48]. Moreover, GAs can be designed with different selection, crossover, and mutation strategies to suit specific problem requirements. In this work, we develop problem-specific crossover and mutation operations, tailored to satisfy the defined constraints of our workflow offloading problem, such as data dependency, the workflow deadline, memory, and security constraints. These operators ensure that offspring solutions are feasible.

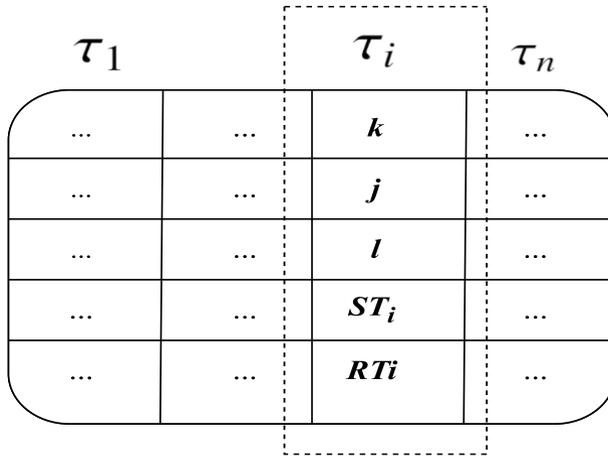
Inspired by the proposed mathematical model, WSEFC-GA is concerned with finding feasible solutions such that the monetary cost of executing a workflow is minimized. Chromosome (solution) representation, population initialization, fitness evaluation, and application of evolutionary operators including selection, crossover, and mutation are the key elements of the proposed WSEFC-GA algorithm. In addition, for each chromosome in the initial population and offspring chromosomes, we evaluate whether the solution is feasible or not. A solution is considered feasible if and only if it satisfies the constraints specified in Sect. 4. The following subsections provide a detailed description of the WSEFC-GA algorithm and its key elements.

### 5.1 Chromosome structure

As mentioned in Sect. 4, a schedule  $\Phi$ , i.e., a solution, is a set of  $n$  assignments, i.e.,  $\Phi = \{A_i \mid i = 1, \dots, n\}$ . Where the assignment  $A_i$  is represented as follows:

$$A_i := \langle \tau_i, N_{kjl}, ST_i, RT_i \rangle$$

where assignment  $A_i$  indicates that task  $\tau_i$  is assigned to  $l$ th node of instance type  $j$  in computing layer  $k$ , i.e.,  $N_{kjl}$ . The notations  $ST_i$  and  $RT_i$  indicate the start time and run time of task  $\tau_i$  on the assigned node, respectively. According to this definition, the representation of a chromosome (solution) in the proposed genetic algorithm is shown in Fig. 3.



**Fig. 3** Chromosome structure

It is a structure of length  $n$ , consisting of cells and the index of the cells indicate the task indexes ( $\tau_i$ ). Each cell consists of the computing layer index ( $k$ ), the instance type index ( $j$ ), the node index ( $l$ ), the start time of the task ( $ST_i$ ), and the run time of the task ( $RT_i$ ) on the assigned node. Since there are data dependencies among tasks, the start time and run time of tasks are included in the chromosome to efficiently manage these dependencies and reduce the computational overhead of the scheduling algorithm, inspired by dynamic programming.

## 5.2 Fitness function

A genetic algorithm uses a fitness function to evaluate the superiority of chromosomes and to determine the evolution of the next generations. To evaluate the superiority of the chromosome  $\Phi$  generated by the proposed WSEFC-GA algorithm, we define the fitness function  $F : \Phi \rightarrow \mathbb{R}$ , where  $F$  assigns a real value to the chromosome  $\Phi$ . Similar to the objective function, in Eq. (16), the function  $F$  calculates the costs of executing a workflow including the cost of renting nodes and the data transfer costs. The fitness function is formulated in Eq. (19).

$$\begin{aligned}
 F = & \sum_{A_i \in \Phi} pr_{kj} \cdot \left[ \frac{\max\{(ST_i + RT_i)\} - \min(ST_i)}{\Delta_t} \right] \\
 & + \sum_{i \in T - \{t1\}} \sum_{t \in pre_i} d_{it} \cdot \theta_{it} \cdot dtc
 \end{aligned} \tag{19}$$

### 5.3 WSEFC-GA algorithm

The procedure of the proposed WSEFC-GA algorithm is shown in Algorithm 1. It starts by assigning algorithm parameters including crossover and mutation rates, maximum number of iterations (MI), and other parameters, in lines 3-5. Then the initial population is generated by the Initial Population-WSEFC algorithm, in line 6, presented in Algorithm 2. If the scheduling problem is not feasible, Algorithm 2 sets  $Feasible = False$ , and the WSEFC-GA algorithm ends, in line 8. Otherwise, Algorithm 2 generates a population of  $PS$  chromosomes such that security, memory, deadline, and precedence constraints are satisfied. For each feasible chromosome, its fitness value is calculated according to the fitness function defined in Eq. (19). After generating the initial population, evolution operators are repeatedly performed by Algorithms 4 and 5 to generate feasible offspring chromosomes, in lines 11-16. After each iteration, the  $PS$  best chromosomes are selected according to the fitness values. Finally, the algorithm returns the best chromosome, i.e., the best schedule  $\Phi$ , based on the computed fitness values, in line 19.

#### Algorithm 1 WSEFC-GA

- 
- 1: **Input:** Tasks list  $T$ , dependency matrix  $D$ , characteristics of provided nodes by computing Layers in  $CL$ .
  - 2: **Output:** The best schedule  $\Phi$
  - 3: Set crossover and mutation rates
  - 4: Set population size,  $PS$ , max number of iteration,  $MI$ .
  - 5: Set  $Feasible = True$ .
  - 6: Generate and evaluate initial population of  $PS$  chromosomes,  $C_i$  ( $i = 1, \dots, PS$ ), via *Algorithm 2*
  - 7: **if**  $Feasible = False$  **then**
  - 8:     Exit;
  - 9: **else**
  - 10:      $iter = 0$
  - 11:     **while**  $iter \leq MI$  **do**
  - 12:         Apply crossover operator and add offspring chromosomes to the *Population* via *Algorithm 4*
  - 13:         Apply mutation operator and add offspring chromosomes to the *Population* via *Algorithm 5*
  - 14:         Select the  $PS$  best chromosomes of *population* based on fitness function in *Eq. (19)*
  - 15:     Increment  $iter$
  - 16:     **end while**
  - 17:      $\Phi$ =The best chromosome based on fitness function in *Eq.(19)*
  - 18: **end if**
  - 19: **return**  $\Phi$ ;
-

## 5.4 Initial population

Since workflow offloading is a complex problem, generating a chromosome by selecting a random computing resource, e.g., computing layer, and instance type, leads to infeasible solutions. Therefore, we propose a problem-specific initial population algorithm that generates feasible chromosomes, shown in Algorithm 2. To improve the performance of the proposed WSEFC-GA algorithm, we integrate the stochastic nature of generating the initial population with heuristic rules to satisfy the security and memory constraints. This hybrid approach helps to find feasible solutions and accelerate the convergence of the algorithm. Inspired by the problem formulation, in line 7, the proposed algorithm selects a computing layer  $k$  for each task so that its security requirement, described in Eq. (1), is satisfied. Then, list  $M$  of instance types provided by  $CL_k$  that satisfy the memory constraint, described in Eq. (3), is created. If  $M$  is not empty, instance type  $j$  is randomly selected from list  $M$ ; then, node  $l$  is randomly selected from node pool  $NP_{kj}$ , in lines 9-11. If  $M$  is empty and  $ts_i = 1$ , the memory constraint cannot be satisfied and the problem is not feasible; in this case, the algorithm set  $Feasible = False$  and it ends, in lines 13-15. If  $M$  is empty and  $ts_i \neq 1$ , the algorithm tries to find an appropriate computing layer for the task, line 17. After selecting computing nodes for all tasks, the run time and the start time of the tasks are calculated via Algorithm 3, in line 21. The deadline constraint is checked, in line 22, when a chromosome is generated. If this constraint is satisfied, the fitness of the chromosome is calculated, and it is added to the population. This procedure is repeated until the maximum value of the iteration is reached. When the algorithm is finished, it checks to see if feasible chromosomes are generated. If  $IntiPop = \emptyset$ , the problem is not feasible and it sets  $Feasible = False$ ; otherwise, it returns the  $IntiPop$ .

The chromosome structure in the proposed algorithm is designed to include the start time and run time of tasks. As mentioned, Algorithm 3 calculates the run time and start time of tasks. This algorithm calculates the run time of each task on the assigned node to it, in line 3. Then it calculates the data transfer time between dependent tasks according to the identified computing layers and nodes, in line 10. Then it computes the ready time of dependent tasks and their start time so that the precedence and the assignment dependency constraints, described in Sect. 4.5, are satisfied, lines 11-12. By integrating start time and run time, the algorithm can efficiently manage and satisfy data dependencies among tasks within a workflow, avoiding redundant calculations, and reducing the computational overhead of the scheduling algorithm.

**Algorithm 2** Initial Population-WSEFC

---

```

1: Input:  $T, D, CL$ , and population size  $PS$ 
2: Result: Initial population  $IntiPop$ 
3:  $IntiPop = \emptyset, Iter = 0$ 
4: while  $Iter \leq PS$  do
5:     Generate chromosome  $C$ 
6:     for all task  $\tau_i$  in  $C$  do
7:         Choose computing layer  $k$  according to Eq. (1)
8:         Create list  $M$  of instance types provisioned by  $CL_k$  satisfying memory constraint
           in Eq. (3);
9:         if  $M \neq \emptyset$  then
10:            Choose instance type  $j$  randomly from list  $M$ ;
11:            Choose node  $l$  randomly from node pool  $NP_{kj}$ ;
12:         else
13:             if  $ts_i = 1$  then
14:                  $Feasible = False$ ;
15:                 Exit;
16:             else
17:                 Goto line 7;
18:             end if
19:         end if
20:     end for
21:     Compute  $RT$  and  $ST$  via Algorithm 3
22:     Check deadline constraint in Eq.(15)
23:     if  $C$  is feasible then
24:         Evaluate  $C$  via fitness function in Eq.(19)
25:         Add  $C$  to  $IntiPop$ 
26:     else
27:         Discard  $C$ 
28:     end if
29:     Increment  $Iter$ 
30: end while
31: if  $IntiPop = \emptyset$  then
32:      $Feasible = False$ 
33: else
34:     return  $IntiPop$ ;
35: end if

```

---

**5.5 Crossover operator**

Selection, crossover, and mutation operators are important components of a genetic algorithm, which are utilized to generate offspring solutions. As mentioned

above, workflow offloading is a complex problem, and proposing problem-specific crossover and mutation operators is a crucial requirement for producing feasible offspring solutions. The crossover operator is used to inherit some chromosome fragments from excellent individuals to generate the next generations. The proposed crossover operator is shown in Algorithm 4.

In this algorithm, we use K-way tournament selection for parent selection, in line 7. It selects K chromosomes from the population randomly and selects the best out of these, according to the fitness function, to become parent  $P1$ . The same process is repeated for selecting parent  $P2$ .

### Algorithm 3 RTST-WSEFC algorithm

---

```

1: Input:  $T, D, CL$  and Chromosome  $C$  .
2: for all task  $\tau_i$  in  $C$  do
3:   Compute  $RT_i$  via Eq.(6)
4: end for
5: for all task  $\tau_i$  in  $C$  do
6:   if  $i = 1$  then
7:      $ReadyT_i = 0$ 
8:   else
9:     for all  $t$  in  $pre_i$  do
10:      Compute  $\beta_{t,i}$  via Eq. (12)
11:      Compute  $ReadyT_i$  via Eq. (8)
12:      Compute  $ST_i$  via Eq. (14)
13:     end for
14:   end if
15: end for

```

---

Then, we apply uniform crossover to generate feasible solutions for the workflow offloading problem, in line 9. The uniform crossover is applied only for changing the computing layer, instance type, and node indexes. As described in Sect. 5.4, for each task, we select a computing layer and an instance type that satisfies the security and memory constraints. Based on the structure of the chromosome and proposed uniform crossover, these constraints are satisfied for offspring solutions. Then, the run time and the start time of tasks are calculated via Algorithm 3 to ensure the precedence and assignment dependency constraints, in line 12. For each generated offspring solution, it checks to see if the deadline constraint is satisfied, in line 13. If this constraint is satisfied, the fitness of the chromosome is calculated, and it is added to the population.

**Algorithm 4** CrossOver-WSEFC

---

```

1: Input:  $T, D, CL, PS$ , and crossover rate.
2: Output: offspring solutions  $sols$ 
3:  $C_{Iter} = PS * crossover\ rate$ 
4:  $Solutions\ sols = \emptyset$ ;
5:  $iter = 0$ 
6: while  $iter \leq C_{Iter}$  do
7:   Select two parents  $P1$  and  $P2$  based on K-way tournament selection
8:   for  $M$  points in  $P1$  do
9:     Apply uniform crossover on  $P1$  and  $P2$  and generate offspring solutions
10:   end for
11:   for all  $C$  in offspring solutions do
12:     Compute  $RT$  and  $ST$  via Algorithm 3
13:     Check deadline constraint in Eq.(15)
14:     if  $C$  is feasible then
15:       Evaluate  $C$  via fitness function in Eq.(19)
16:       Add  $C$  to  $sols$ 
17:     else
18:       Discard  $C$ 
19:     end if
20:   end for
21:   Increment  $iter$ 
22: end while
23: return  $sols$ 

```

---

**5.6 Mutation operator**

The mutation operator in GA replaces some gene values with other values in the gene domain to improve population diversity. Similar to the crossover operator detailed above, we propose a problem-specific mutation operator for the scheduling problem, shown in Algorithm 5. This algorithm uses K-way tournament selection for selecting parent  $P$  from the current population, in line 7. Then, it selects some mutation points (tasks) randomly from the selected parent  $P$ . For each selected mutation point the following process is performed: If the task of the selected point is private, i.e.,  $ts_i = 1$ , then the algorithm continues without performing the mutation operator since the computing resource cannot be changed for the private task, line 11. Otherwise, for the selected mutation point, three mutation operators are performed on the computing layer, instance type, and node indexes, respectively, in lines 13- 21. Similar to the initial population algorithm detailed above, the mutation operators are performed so that the security and memory requirements of the tasks are satisfied. After applying the mutation operators on the selected points, the run time and start time of tasks are calculated via Algorithm 3 to satisfy the precedence and assignment dependency constraints, in line 22. Then, it checks if the deadline

constraint is satisfied, in line 23. If this constraint is satisfied, the fitness of the new chromosome is calculated, and it is added to the population.

### Algorithm 5 Mutation-WSEFC

---

```

1: Input:  $T, D, CL, PS$ , and mutation rate.
2: Output: offspring  $sols$ 
3:  $M_{Iter} = PS * \text{mutation rate}$ 
4:  $Solutions\ sols = \emptyset$ ;
5:  $iter = 0$ 
6: while  $iter \leq M_{Iter}$  do
7:   Select parent  $P$  based on K-way tournament selection
8:   Select  $U$  mutation points (tasks) randomly from the chosen parent  $P$ 
9:   for all  $\tau_i$  in the selected points do
10:    if  $ts_i = 1$  then
11:      break;
12:    else
13:      Choose computing layer  $k$  according to Eq. (1)
14:      Create list  $M$  of instance types provisioned by  $CL_k$  satisfying memory
      constraint in Eq. (3);
15:    end if
16:    if  $M \neq \emptyset$  then
17:      Choose instance type  $j$  randomly from list  $M$ ;
18:      Choose node  $l$  randomly from node pool  $NP_{kj}$ ;
19:    else
20:      Goto line 13;
21:    end if
22:    Compute  $RT$  and  $ST$  via Algorithm 3
23:    Check deadline constraint in Eq.(15)
24:    if  $P$  is feasible then
25:      Evaluate offspring chromosome  $C$  via fitness function in Eq.(19)
26:      Add  $C$  to  $sols$ 
27:    else
28:      Discard  $C$ 
29:    end if
30:  end for
31:  Increment  $iter$ 
32: end while
33: return  $sols$ 

```

---

## 5.7 Time complexity

The problem of task scheduling is an NP-hard problem. Therefore, as the problem size increases, the computational complexity of solving the problem with a mathematical programming model increases rapidly. The time complexity of a genetic algorithm depends on the genetic operators, the representation of the chromosomes, and the population size. Given the usual choices (one-point mutation, one-point crossover, selection), the time complexity of the WSEFC-GA algorithm is

**Table 3** Instance type specifications provided by the edge-cloud computing

Computing layer	Instance type	Price (\$/hour)	Performance (MIPS)	Memory (GB)
Cloud nodes	Type 1	0.2	3000	4
	Type 2	0.24	3500	8
	Type 3	0.42	5000	16
	Type 4	0.48	5500	32
	Type 5	0.62	7500	64
Fog nodes	Type 1	0.11	1500	4
	Type 2	0.14	2200	8
	Type 3	0.16	2500	16
Edge node	Type 1	0	1000	2

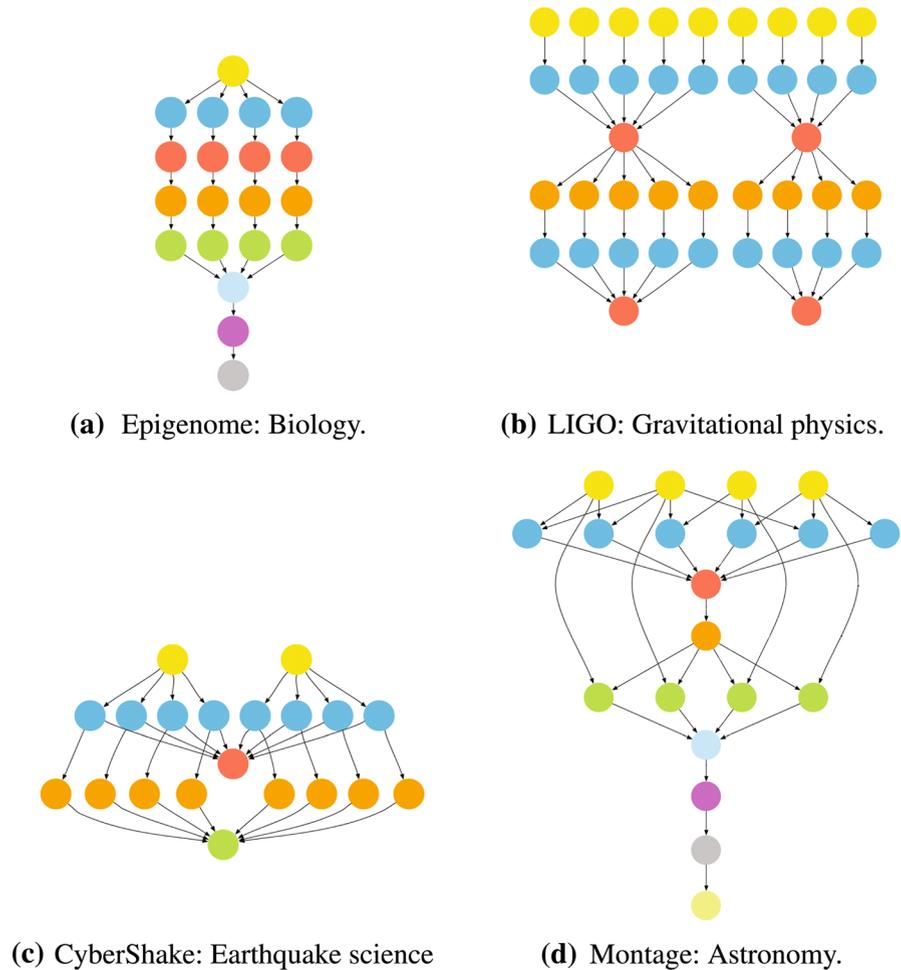
$O(g(nm + nm + n))$ , where  $g$  is the number of generations,  $n$  is the population size, and  $m$  is the size of the individuals.

## 6 Experimental results

To evaluate the performance of the proposed mathematical model (MPM), we linearized the model's constraints. Furthermore, we solved the model as a quadratic objective model with IBM ILOG CPLEX Optimization Studio version 22.1. The proposed genetic algorithm (WSEFC-GA) is also implemented in MATLAB. Specifically, experiments are performed on a PC with Intel Core i7 2.3 GHz, 32 GB RAM, and a Windows 10 operating system.

### 6.1 Experimental settings

Specifications of node instances provided by edge, fog, and cloud layers are listed in Table 3. The specifications of cloud nodes are based on Amazon EC2 instance types, and the cloud layer provides more powerful node instances than the fog layer. We assumed that fog servers also use virtualization technology to create and manage nodes [49, 50]. We considered different node instances in cloud and fog layers, each with a specific memory capacity and performance level. However, in the experiments, we considered limited computing capacity in the edge and fog layers. We consider one node in an edge device. In the fog layer, we consider 3 node instances, each of which has 5 nodes available for scheduling a single workflow. Since providing computing nodes for a fog server and cloud provider incurs costs, such as power and maintenance expenses, running nodes in the fog/cloud layers are charged. The average bandwidth between two different computing layers is set to 10 Mbps and the average bandwidth between two different nodes in the same computing layer is set to 100 Mbps. The data transmission cost is considered  $d_{tc} = 0.02$  (\$ per GB).



**Fig. 4** The structure of some scientific workflows [51]

To investigate the behavior of the MPM and the WSEFC-GA, we evaluate the cost of executing some real-world workflows such as the Epigenome, LIGO, Montage, and CyberShake workflows with small workflow sizes (20 tasks) and large workflow sizes (100 tasks). The structures of these workflows are shown in Fig. 4. Epigenome workflow is used in the domain of biology, and it is involved in mapping the epigenetic state of human cells on a genome-wide scale. The LIGO workflow involves several key stages to detect and analyze gravitational waves. CyberShake is a computational workflow designed to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. The Montage workflow is designed to create large-scale astronomical image mosaics. Since different workflows of wide-ranging domains are frequently executed by different research groups

or companies on cluster or cloud systems, their characteristics (e.g., structure, task execution time, data dependencies, memory, input/output data size) are known beforehand [41, 42, 51, 52]. The work in [51] developed a set of workflow profiling tools called wfprof to provide detailed information about the various computational tasks present in a workflow. Moreover, the characteristics of these workflows (e.g., structure, task execution time, data dependencies, memory, input/output data size) are available from the Pegasus workflow generator.<sup>1</sup>

In this section, we have conducted two types of experiments. Some experiments evaluate the performance of the proposed genetic algorithm, WSEFC-GA, and the proposed mathematical model, MPM, for small workflows, since MPM is time-consuming for large problem instances. In other experiments, we compared the WSEFC-GA algorithm with a heuristic GA-based (HGA) algorithm [28], a PSO-based technique [35], and a Bee Life Algorithm (BLA) [27] for large workflows. These algorithms propose deadline-constrained and cost-optimized task scheduling in the literature. Moreover, we implement a constraint satisfaction algorithm (CSA) that finds feasible solutions that satisfy all the constraints mentioned in Sect. 4 for the workflow scheduling problem without applying any optimization criterion.

- HGA [28] proposes a task scheduling algorithm for bag-of-tasks applications in fog-cloud computing. The algorithm formulates task scheduling as a permutation-based optimization problem, using a modified genetic algorithm to generate permutations for arriving tasks at each scheduling round. For each proposed permutation, the tasks are assigned in the given order to VMs, which offers the minimum expected finish time. Additionally, the chosen VMs must have sufficient memory capacity to accommodate tasks. The objective function of this scheduling algorithm is to optimize the total execution cost while satisfying the deadline constraint.
- The objective of the PSO [35] is to generate workflow scheduling in order to minimize the cost and meet the workflow deadline. PSO relies on the exchange of information between individuals, called particles, within a population, called a swarm. Each particle updates its trajectory based on its own previous best position (local best) and the best position achieved by the entire swarm (global best).
- The BLA algorithm [27] minimizes the cost of processing tasks while considering the memory requirements of tasks. It assumes that several tasks can be assigned to the same fog node and identifies the order of executing tasks on the assigned nodes. The BLA algorithm starts by randomly generating an initial population and uses crossover and mutation operators to generate offspring solutions. For generating offspring solutions, the population is sorted according to the fitness function, with the top individual selected as the queen, the following  $D$  bees classified as drones. The queen mates with a group of drones to produce offspring using crossover and mutation operators.

---

<sup>1</sup> <https://pegasus.isi.edu/>

**Table 4** WSEFC-GA, BLA, PSO parameters

	BLA			WSEFC-GA	PSO
Population size	Queen: 1	Drones: 30	Workers: 69	100	100
Number of generations	500			500	500
Crossover rate	85%			85%	$c1 = c2 = 1.5$
Mutation rate	1%			1%	$w = 0.9 \rightarrow 0.1$

Since the compared PSO, BLA, and HGA algorithms did not consider security requirements in the task offloading problem, in the experiments in which WSEFC-GA is compared with these algorithms, all tasks are considered public tasks. In the experiments, the population size is 100 while the number of generations is 250. The crossover and mutation rates are 85% and 1%, respectively. Parent selection is based on K-way tournament selection which  $K = 3$  in the experiments. We also normalized the execution cost of a workflow by dividing its execution cost by  $cost_{min}$  since various workflows have different characteristics. Table 4 indicates the parameter settings of the evolutionary algorithms.

## 6.2 Results

Figure 5 shows the normalized cost over changes in the deadline for small Epigenome workflow. This workflow is used in the domain of Biology and its structure is depicted in Fig. 4a. We performed experiments for this workflow over changes in the deadline. Tasks of this workflow are compute-intensive, and all tasks are considered public tasks. It can be seen from Fig. 5, with the increase in the deadline the cost decreases since in tight deadlines more powerful node instances from fog or cloud are selected which are more expensive. Notably, solutions generated by WSEFC-GA show some variance from the optimal solutions of MPM particularly when deadlines are tight. Mathematical programming models like MPM find globally optimal solutions even in complex constrained scenarios such as those with tight deadlines. However, as the deadline extends, WSEFC-GA demonstrates an ability to produce solutions that closely approximate the optimal solutions

A comparison among the normalized cost for WSEFC-GA, PSO, BLA, HGA, and CSA for large Epigenome workflow is depicted in Fig. 6. In this experiment, the problem is feasible for deadlines greater than  $D^w \geq 0.2$  (h). Furthermore, WSEFC-GA finds more cost-effective solutions compared to other algorithms, especially when the deadline is tight, i.e., the problem becomes more complex. While the HGA algorithm generally outperforms BLA and PSO, all these algorithms outperform CSA, which can only produce feasible solutions.

Figure 7 indicates the normalized cost over changes in the data size of tasks for the Montage workflow. The structure of the Montage workflow is shown in Fig. 4d. This workflow processes an image and we performed the experiments with different values for the image size, namely  $DS = \{1, \dots, 10\}$  GB. In this

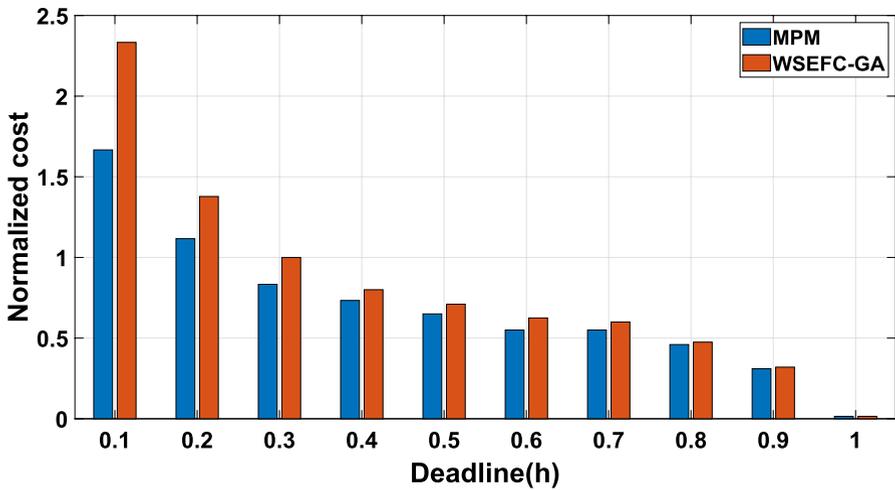


Fig. 5 Normalized cost over changes in the deadline for small Epigenome workflow

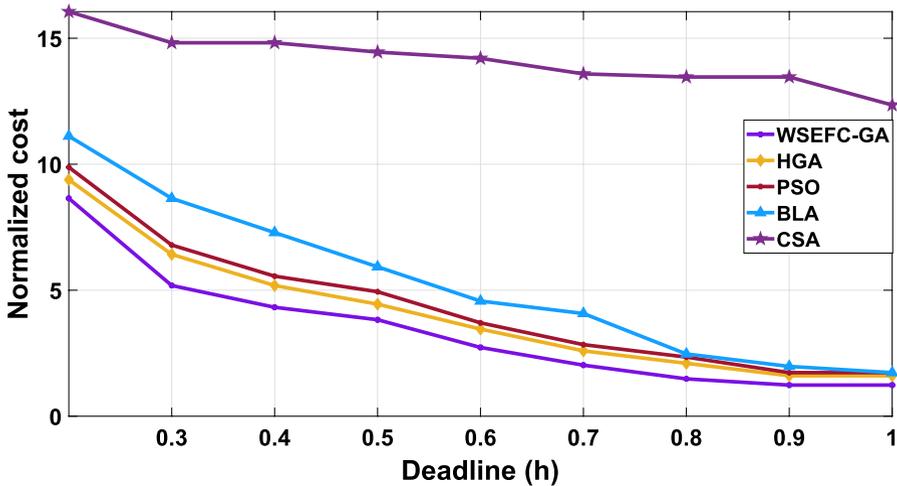


Fig. 6 Normalized cost over changes in the deadline for large Epigenome workflow

experiment, the deadline is fixed, i.e.,  $D^w = 0.4 (h)$ , and the edge device cannot perform all the workflow tasks.

Figure 7 demonstrates that the normalized cost of WSEFC-GA and MPM increases with the increase in the data size (DS). With an increase in DS, the data transfer time also increases, necessitating the selection of more powerful node instances to meet the workflow deadline. It is observed that WSEFC-GA achieves solutions close to the optimal cost. However, as the data size increases,

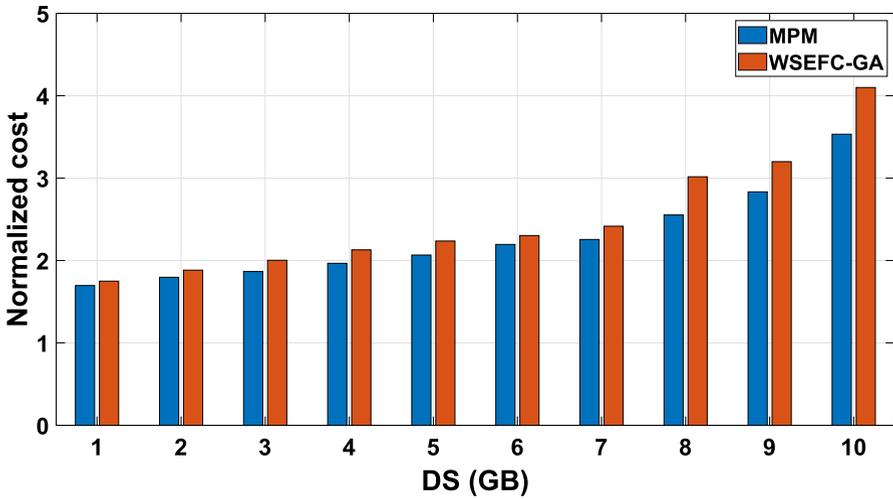


Fig. 7 Normalized cost over changes in the data size for small Montage workflow

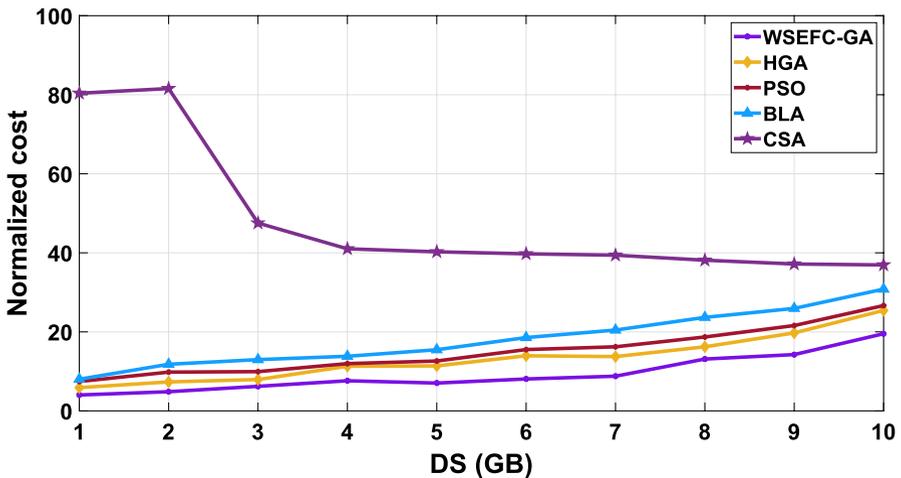


Fig. 8 Normalized cost over changes in the data size for large Montage workflow

WSEFC-GA solutions diverge somewhat from the optimal cost, indicating increased complexity in addressing the problem.

Figure 8 shows the comparison of normalized cost for WSEFC-GA, PSO, BLA, HGA, and CSA for a large Montage workflow. WSEFC-GA finds cost-effective solutions compared to PSO, HGA, and BLA. The CSA algorithm gave the worst results among the algorithms. The cost-decreasing trend for the CSA algorithm with increasing data size is due to the narrowing set of feasible solutions. As the data size increases, the data transfer time increases, and spreading dependent tasks across different computing nodes leads to infeasible solutions.

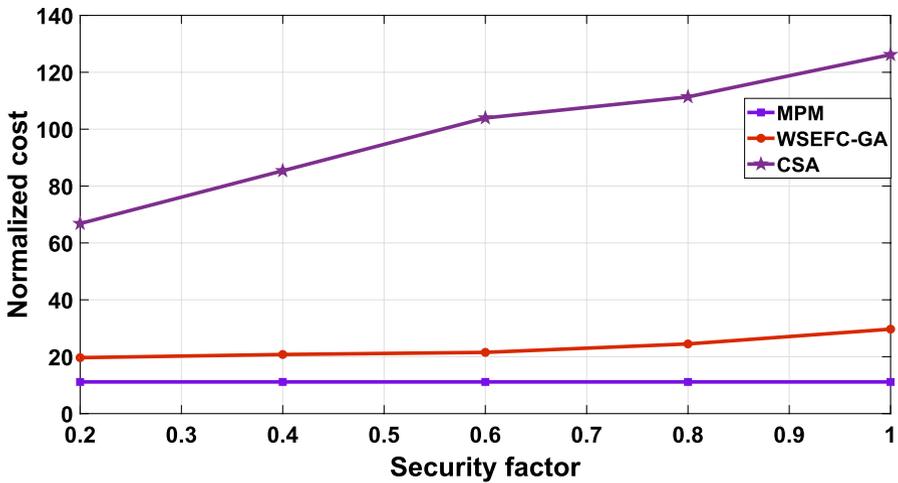


Fig. 9 Normalized cost over changes in the data size for CyberShake workflow

Figure 9 shows the impact of security requirements on the normalized cost for small CyberShake workflow. As mentioned, when a task is private, it can only be executed in the edge layer while a public task can be executed in the edge, fog, or cloud layer. To perform this experiment, we considered a scenario in which workflow tasks are compute-intensive and executing all the tasks in the edge device is not feasible to meet the deadline. Therefore, the scheduler outsources some or all tasks to the fog or cloud layers to meet the workflow deadline.

In this experiment, when the security factor is 0, all the tasks are private while the security factor 1 indicates that all tasks are public, and for example, security factor = 0.2 indicates that 20 percent of tasks are public. In Fig. 9, when the security factor = 0, the normalized cost is not shown, because executing all tasks in the edge layer are not feasible. As shown, the optimal solutions of MPM are not affected by the changes in the security factor. However, the cost of WSEFC-GA and CSA increases with the increase in the security factor. WSEFC-GA and CSA are random-based approaches and with the increase in security factor, the set of feasible solutions increases since for each public task, a node in the edge, fog, or cloud can be selected. However, WSEFC-GA is much more cost-efficient than CSA.

### 6.3 Convergence Analysis and Running Time

In this subsection, we express the convergence analysis and running time of the MPM, WSEFC-GA, PSO, HGA, and BLA approaches. Figure 10 illustrates the convergence of three evolutionary algorithms, namely WSEFC-GA, PSO, and BLA, applied to the LIGO workflow, tracked over 500 generations. Since HGA is a heuristic algorithm, it is excluded from this experiment. The population was observed every 10 generation, and the average fitness value of the entire population was recorded. Notably, both PSO and BLA exhibited rapid convergence after

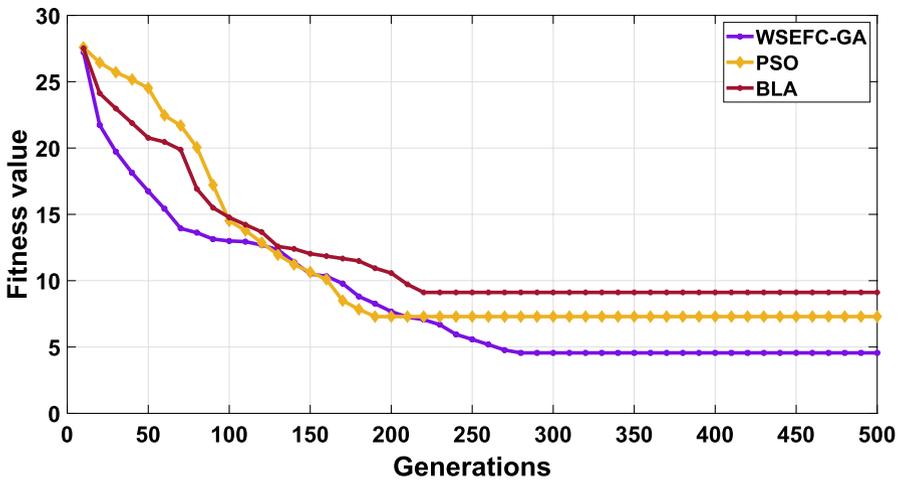


Fig. 10 Convergence comparison between three evolutionary algorithms

**Table 5** Running time of the approaches for the workflows in seconds with different sizes

	Epigenome		LIGO		Montage		Cyber-shake	
Approches	20	100	20	100	20	100	20	100
WSEFC-GA	9.4	124.8	9.8	110.6	9.4	96.6	9.8	98.3
PSO	9.6	128.3	10.3	125.2	10.2	110.7	11.2	113.8
HGA	16.8	178.8	16.3	176.3	15.2	171.2	16.5	173.7
BLA	12.3	143.4	13.4	139.6	11.9	137.8	12.8	139.1

approximately 200 generations, albeit producing solutions with high fitness values compared to those achieved by the WSEFC-GA algorithm with a cost minimization objective. In contrast, WSEFC-GA converged after 250 generations, maintaining population diversity and demonstrating the capability to reach more cost-effective solutions.

Table 5 shows the running time of the approaches for small (20 tasks) and large (100 tasks) instances of Epigenome, Montage, Cybershake, and LIGO workflows. For fair comparisons, the mathematical model is excluded from this experiment, as it was only run for small instances of workflows. It is observed that the computational overhead increases as the number of tasks increases for all algorithms. Moreover, the running time of WSEFC-GA is better than that of the other algorithms because we proposed problem-specific crossover, mutation, and initial population strategies that generate feasible solutions efficiently. Since BLA involves more complex operations, such as recruiting other workers to collect food, it has more computational overhead than the PSO algorithm. HGA uses a permutation-based scheduling algorithm and has the highest computational overhead. Nevertheless, the running time of the proposed WSEFC-GA algorithm is negligible compared to the workflow execution time. These results show that

the PSO algorithm converges faster in terms of iterations, but the proposed GA has a lower computational overhead. This is because the proposed problem-specific crossover, mutation, and initial population operators generate feasible solutions more effectively. Moreover, the chromosome structure in the proposed algorithm is designed to include the start time and run time of tasks. By integrating start time and run time, the algorithm can efficiently manage and satisfy data dependencies among tasks within a workflow, avoiding redundant calculations, and resulting in a faster and more efficient scheduling algorithm.

## 7 Conclusion and future works

This paper addressed the problem of workflow offloading in the edge-cloud computing continuum. It proposed a nonlinear mathematical programming model, called MPM. This model minimizes the monetary cost of workflow execution under security and deadline constraints. This paper also proposed a genetic algorithm, called WSEFC-GA, for the problem of workflow offloading. The WSEFC-GA is proposed with these key insights: (1) The stochastic nature of generating the initial population is combined with heuristic rules to ensure that the problem constraints are satisfied. (2) Problem-specific crossover and mutation operators are proposed to fulfill security and other constraints of the problem when generating offspring chromosomes. (3) It finds near-optimal solutions that minimize the cost of executing workflow. (4) Inspired by dynamic programming, start time and run time of tasks are considered in the chromosome structure to reduce the computational overhead. The proposed model was solved with the CPLEX solver, and the WSEFC-GA was implemented in MATLAB. We also proposed an algorithm that finds feasible solutions without applying any optimization criterion, called CSA. Experimental evaluation shows that MPM can efficiently deal with the data dependency among tasks. The comparison of MPM, WSEFC-GA, PSO, BLA, HGA, and CSA shows that WSEFC-GA can find a good approximation for the optimal solutions in a reasonable time.

In future work, we intend to extend our proposed approach to take into account communication congestion in the workflow offloading problem. As part of our future work, we also plan to extend our proposed approach for scheduling of parallel workflows. This will involve addressing more complex scheduling policies and the associated challenges.

**Acknowledgements** The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the AORTA projects and KKS foundation through the project SEINE. Also, the work is supported by XPRES project.

**Author contributions** Somayeh Abdi was contributed to conceptualization, methodology, software, validation, formal analysis, writing—original draft. Mohammad Asjaei was contributed to conceptualization, methodology, formal analysis, reviewing—manuscript. Saad Mubeen was contributed to conceptualization, methodology, formal analysis, reviewing—manuscript

**Funding** Open access funding provided by Mälardalen University.

## Declarations

**Conflict of interest** All authors declare that they have no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Aazam M, Zeadally S, Harras KA (2018) Offloading in fog computing for IoT: review, enabling technologies, and research opportunities. *Future Gener Comput Syst* 87:278–289
2. Liu B, Meng S, Jiang X, Xu X, Qi L, Dou W (2021) A QoS-guaranteed online user data deployment method in edge cloud computing environment. *J Syst Architect* 118:102185
3. Shakarami A, Shakarami H, Ghobaei-Arani M, Nikoufostar E, Faraji-Mehmandar M (2022) Resource provisioning in edge/fog computing: a comprehensive and systematic review. *J Syst Architect* 122:102362
4. Yousefpour A, Fung C, Nguyen T, Kadiyala K, Jalali F, Niakanlahiji A, Kong J, Jue JP (2019) All one needs to know about fog computing and related edge computing paradigms: a complete survey. *J Syst Architect* 98:289–330
5. Tang F, Liu C, Li K, Tang Z, Li K (2021) Task migration optimization for guaranteeing delay deadline with mobility consideration in mobile edge computing. *J Syst Architect* 112:101849
6. You Q, Tang B (2021) Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *J Cloud Comput* 10:1–11
7. Ren J, Qin T (2023) Decentralized blockchain-based and trust-aware task offloading strategy for healthcare IoT. *IEEE Internet Things J* 11:829–847
8. Zhou H, Zhang Z, Li D, Su Z (2022) Joint optimization of computing offloading and service caching in edge computing-based smart grid. *IEEE Trans Cloud Comput* 11:1122–1132
9. Prasad CR, Kumar S, Rao PR, Kollem S, Yalabaka S, Samala S (2022) Optimization of task offloading for smart cities using IoT with fog computing—a survey. In: 2022 International Conference on Signal and Information Processing (ICONSIP), IEEE, pp 1–5
10. Yeh C, JoG Do, Ko YJ, Chung HK (2023) Perspectives on 6 g wireless communications. *ICT Express* 9(1):82–91
11. Liu Z, Dai P, Xing H, Yu Z, Zhang W (2021) A distributed algorithm for task offloading in vehicular networks with hybrid fog/cloud computing. *IEEE Trans Syst Man Cybern Syst* 52(7):4388–4401
12. Akhlaqi MY, Hanapi ZBM (2023) Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. *J Netw Comput Appl* 212:103568
13. Ometov A, Molua OL, Komarov M, Nurmi J (2022) A survey of security in cloud, edge, and fog computing. *Sensors* 22(3):927
14. Alhroob A, Samawi VW (2018) Privacy in cloud computing: intelligent approach (research poster). In: 2018 International Conference on High Performance Computing and Simulation (HPCS), IEEE, pp 1063–1065
15. Parikh S, Dave D, Patel R, Doshi N (2019) Security and privacy issues in cloud, fog and edge computing. *Proced Comput Sci* 160:734–739
16. Abdi S, Ashjaei M, Mubeen S (2022) Cognitive and time predictable task scheduling in edge-cloud federation. In: 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, pp 1–4

17. Bisikalov OV, Kovtun VV, Kovtun OV, Danylchuk OM (2021) Mathematical modeling of the availability of the information system for critical use to optimize control of its communication capabilities. *Int J Sens Wirel Commun Control* 11(5):505–517
18. Buras N (1985) An application of mathematical programming in planning surface water storage 1. *JAWRA J Am Water Resour Assoc* 21(6):1013–1020
19. Grossmann IE, Guillén-Gosálbez G (2010) Scope for the application of mathematical programming techniques in the synthesis and planning of sustainable processes. *Comput Chem Eng* 34(9):1365–1376
20. Samuels JM (1965) Opportunity costing: an application of mathematical programming. *J Account Res* 3(2):182–191
21. Gen M, Lin L (2023) Genetic algorithms and their applications. In: Springer handbook of engineering statistics, Springer, pp 635–674
22. Islam A, Debnath A, Ghose M, Chakraborty S (2021) A survey on task offloading in multi-access edge computing. *J Syst Architect* 118:102225
23. Saeik F, Avgeris M, Spatharakis D, Santi N, Dechouniotis D, Violos J, Leivadetas A, Athanasopoulos N, Mitton N, Papavassiliou S (2021) Task offloading in edge and cloud computing: a survey on mathematical, artificial intelligence and control theory solutions. *Comput Netw* 195:108177
24. Kumari N, Yadav A, Jana PK (2022) Task offloading in fog computing: a survey of algorithms and optimization techniques. *Comput Netw* 214:109137
25. Nguyen BM, Thi Thanh Binh H, The Anh T, Bao Son D (2019) Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud-fog computing environment. *Appl Sci* 9(9):1730
26. Nan Z, Wenjing L, Zhu L, Zhi L, Yumin L, Nahar N (2022) A new task scheduling scheme based on genetic algorithm for edge computing. *Comput Mater Contin* 71(1):843–854
27. Bitam S, Zeadally S, Mellouk A (2018) Fog computing job scheduling optimization based on bees swarm. *Enterp Inf Syst* 12(4):373–397
28. Abohamama AS, El-Ghamry A, Hamouda E (2022) Real-time task scheduling algorithm for IoT-based applications in the cloud-fog environment. *J Netw Syst Manage* 30(4):54
29. Wang X, Ma L, Li H, Yin Z, Luan T, Cheng N (2022) Digital twin-assisted efficient reinforcement learning for edge task scheduling. In: IEEE 95th Vehicular Technology Conference: (VTC2022-Spring). IEEE 2022, pp 1–5
30. Lou J, Tang Z, Jia W, Zhao W, Li J (2023) Startup-aware dependent task scheduling with bandwidth constraints in edge computing. *IEEE Trans Mobile Comput* 23:1586–1600
31. Liu H, Xin R, Chen P, Zhao Z (2022) Multi-objective robust workflow offloading in edge-to-cloud continuum. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), IEEE, pp 469–478
32. Xu M, Mei Y, Zhu S, Zhang B, Xiang T, Zhang F, Zhang M (2023) Genetic programming for dynamic workflow scheduling in fog computing. *IEEE Trans Serv Comput* 16:2657–2671
33. Saeed A, Chen G, Ma H, Fu Q (2023) A memetic genetic algorithm for optimal IoT workflow scheduling. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, pp 556–572
34. El Haber E, Nguyen TM, Assi C (2019) Joint optimization of computational cost and devices energy for task offloading in multi-tier edge-clouds. *IEEE Trans Commun* 67(5):3407–3421
35. Alsurdeh R, Calheiros RN, Matawie KM, Javadi B (2018) Cloud resource provisioning for combined stream and batch workflows. In: IEEE 37th International Performance Computing and Communications Conference (IPCCC). IEEE 2018, pp 1–8
36. Liu L, Chang Z, Guo X, Mao S, Ristaniemi T (2017) Multiobjective optimization for computation offloading in fog computing. *IEEE Internet Things J* 5(1):283–294
37. Chen S, Chen B, Tao X, Xie X, Li K (2022) An online dynamic pricing framework for resource allocation in edge computing. *J Syst Architect* 133:102759
38. Bharathi S, Chervenak A, Deelman E, Mehta G, Su M-H, Vahi K (2008) Characterization of scientific workflows, in, third workshop on workflows in support of large-scale science. *IEEE* 2008:1–10
39. Openalpr automatic license plate recognition. <http://www.openalpr.com>. Accessed 22 Dec 2020
40. Wu H, Knottenbelt W, Wolter K, Sun Y (2016) An optimal offloading partitioning algorithm in mobile cloud computing. In: International Conference on Quantitative Evaluation of Systems, Springer, pp 311–328

41. Aceto L, Morichetta A, Tiezzi F (2015) Decision support for mobile cloud computing applications via model checking. In: 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, IEEE, pp 199–204
42. Salaht FA, Desprez F, Lebre A (2020) An overview of service placement problem in fog and edge computing. *ACM Comput Survays CSUR* 53(3):1–35
43. Qin S, Pi D, Shao Z (2022) Ails: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment. *Expert Syst Appl* 198:116824
44. Du J, Leung JY-T (1989) Complexity of scheduling parallel task systems. *SIAM J Discret Math* 2(4):473–487
45. Ibrahim M, Nabi S, Hussain R, Raza MS, Imran M, Kazmi SA, Oracevic A, Hussain F (2020) A comparative analysis of task scheduling approaches in cloud computing. In: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE 2020, pp 681–684
46. Michael LP (2018) Scheduling: theory, algorithms, and systems. Springer
47. Huang J, Hu H (2024) Hybrid beluga whale optimization algorithm with multi-strategy for functions and engineering optimization problems. *J Big Data* 11(1):3
48. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. *Adv Eng Softw* 69:46–61
49. Caprolu M, Di Pietro R, Lombardi F, Raponi S (2019) Edge computing perspectives: architectures, technologies, and open security issues. In: 2019 IEEE International Conference on Edge Computing (EDGE), IEEE, pp 116–123
50. Varghese B, Reano C, Silla F (2018) Accelerator virtualization in fog computing: Moving from the cloud to the edge. *IEEE Cloud Comput* 5(6):28–37
51. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. *Futur Gener Comput Syst* 29(3):682–692
52. De Maio V, Brandic I (2018) First hop mobile offloading of dag computations. In: 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE 2018, pp 83–92

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.