Journal of Parallel and Distributed Computing 202 (2025) 105069

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

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

A knowledge-driven approach to multi-objective IoT task graph scheduling in fog-cloud computing

Hadi Gholami, Hongyang Sun *

Department of EECS, University of Kansas, Lawrence, KS, USA

ARTICLE INFO

Keywords:
Cloud computing
Fog computing
Genetic algorithm
Multi-objective optimization
Random forest
Task graph scheduling

ABSTRACT

Despite the significant growth of Internet of Things (IoT), there are prominent limitations of this emerging technology, such as limited processing power and storage. Along with the expansion of IoT networks, the fog-cloud computing paradigm has been developed to optimize the provision of services to IoT users by offloading computations to the more powerful processing resources. In this paper, with the aim of optimizing multiple objectives of makespan, energy consumption, and cost, we develop a novel automatic three-module algorithm to schedule multiple task graphs offloaded from IoT devices to the fog-cloud environment. Our algorithm combines the Genetic Algorithm (GA) and the Random Forest (RF) classifier, which we call Hybrid GA-RF (HGARF). Each of the three modules has a responsibility and they are repeated sequentially to extract knowledge from the solution space in the form of IF-THEN rules. The first module is responsible for generating solutions for the training set using a GA. Here, we introduce a chromosome encoding method and a crossover operator to create diversity for multiple task graphs. By expressing a concept called bottleneck and two conditions, we also develop a mutation operator to identify and reduce the workload of certain processing centers. The second module aims at generating rules from the solutions of the training set, and to that end employs an RF classifier. Here, in addition to proposing features to construct decision trees, we develop a format for extracting and recording IF-THEN rules. The third module checks the quality of the generated rules and refines them by predicting the processing resources as well as removing less important rules from the rule set. Finally, the developed HGARF algorithm automatically determines its termination condition based on the quality of the provided solutions. Experimental results demonstrate that our method effectively improves the objective functions in large-size task graphs by up to 13.24 % compared to some state-of-the-art methods.

1. Introduction

The Internet of Things (IoT) ecosystem is becoming ubiquitous with the increasing developments of web-based smart device sensors and communication hardware to collect, process and send data. By bringing the ability to access data and information in any place at any time, IoT is helping to manage people's lives and work in an unprecedented way. Moreover, by serving requests automatically, it has also improved the quality of business and services and reduced the need for human intervention. Therefore, IoT is increasingly being used with massive amounts of diverse data generated by millions of users [1]. One type of data generated by IoT users is graph data, which has a wide range of applications in scientific fields (i.e., bioinformatics, physics, and earthquakes) and smart cities. Task graph applications, which consist of a set of tasks with edges that show the dependency (i.e., data flow) between them, are typically described as Directed Acyclic Graphs (DAGs) [2]. The increased rate of data generation by IoT users, especially in task graph applications with concurrent structure, poses a challenge for data management and processing. Considering the limitations of IoT devices in their processing power, storage, and energy, offloading requests to cloud computing environment is often carried out to meet the Quality of Service (QoS) requirement.

Cloud computing with a centralized processing paradigm has high processing and storage capacity, which utilizes large Data Centers (DCs) to provide IoT users with ubiquitous access for resource sharing and supply [3]. Although virtualization mechanism in the cloud results in the support of IoT development, the concentration of resources in cloud environments causes the division of resources among IoT users and the cloud, which entails network delay [4]. Delay degrades the QoS and is detrimental to location-aware and delay-sensitive tasks (e.g., medical

* Corresponding author.
E-mail addresses: hadi.gholami@ku.edu (H. Gholami), hongyang.sun@ku.edu (H. Sun).

https://doi.org/10.1016/j.jpdc.2025.105069
Received 16 February 2024; Received in revised form 4 October 2024; Accepted 17 March 2025
Available online 18 March 2025

activities) [5]. While delay-tolerant tasks can be processed in the cloud, high bandwidth and unreliable and inconsistent network access are other challenges of this technology. Therefore, recent years have witnessed fog computing as a way to enhance the QoS and the processing capacity of IoT requests. With Micro Data Centers (MDCs) that possess moderate processing capability, the fog has a decentralized structure, and its resources are abundantly available geographically to serve local requests in their regions. Owing to its hardware that occupies smaller space, the fog is closer to IoT devices and accelerates data aggregation, processing, and storage. As such, the fog contributes to reducing the time required for requests to reach resources, thereby increasing response time [6]. It is evident that a combination of cloud and fog technology, or the so-called fog-cloud computing, can respond to a massive flow of requests generated by IoT devices and end-users. However, deciding on the distribution of requests among processing stations in the cloud or fog and scheduling resources effectively remain difficult tasks.

From the perspective of fog-cloud providers, minimizing the energy consumption of MDCs/DCs (from processors, cooling systems, power supply systems, etc.) is also important, because failure to manage this leads to increased financial costs and reduced profits. While hardware-based techniques such as Dynamic Power Management (DPM) [7] and Dynamic Voltage and Frequency Scaling (DVFS) [8] have been presented for energy efficiency, studies on the development of software-based methods such as scheduling algorithms have also been shown as effective solutions. From the perspective of IoT users, minimizing the economic cost is one of their preferences too. Although IoT users benefit from the processing, storage and communication capabilities of the fog-cloud, the cost-effectiveness of this offloading should be carefully considered. Thus, leveraging fog-cloud processing capabilities to reduce processing time needs to be balanced with the offloading costs; logically, using services offered by more advanced equipment leads to an increase in cost but a decrease in execution time, and vice versa. There is a need for a good trade-off between the cost for the use of fog-cloud resources and the application execution time.

Thus, an efficient scheduler should propose a suitable solution considering the objectives of both fog-cloud providers and IoT users. In addition, the scheduler's performance becomes more prominent when it receives multiple task graphs from multiple IoT devices per its scheduling time slot. Although different prioritization policies can be adopted, these policies alone cannot be a solution when facing multiple large-scale task graphs. Hence, batch placement of task graphs is another issue that affects the performance of the scheduler to improve the objective functions. To achieve the goal of developing a good scheduling solution, we argue that it is necessary to consider the following three aspects.

The first aspect is considering the required arrangements for learning-based methods. The use of intelligent planning in scheduling problems has become widespread due to the ability to deduce rules via the exploration and exploitation process [9]. In particular, the current trend toward task graph offloading for IoT devices has led to increasing employment of learning-based algorithms in the fog-cloud environments [10–12]. While machine learning algorithms with their continuous refinements can lead to increased accuracy and efficiency in decision-making, it should be noted that they as data-driven models are designed to address low-complexity and high-uncertainty problems [13]. Although the high-uncertainty aspect includes the current state of this problem, it should be kept in mind that the problem is NP-hard and inherently of high-complexity [14]. Therefore, it is necessary to use a method that, in addition to decentralizing the presentation of the solution in the problem space, extracts rules so that better solutions can be predicted by delving into the rules. In other words, since fog-cloud providers have MDCs/DCs with different characteristics and at a distance from each other, adopting a single rule for all MDCs/DCs may not be suitable; rules are needed to take into account different conditions.

The second aspect is on the design of an efficient method for allocating resources. Developing a method for task graph scheduling often involves two phases: Task Prioritizing Phase (TPP) and Processor Selection Phase (PSP) [15]. In TPP, tasks are prioritized based on their significance, and in PSP, tasks are assigned to MDCs/DCs based on rules. For TPP, many algorithms have been presented in the literature, among which intelligent algorithms have been particularly helpful for better understanding the problem space and providing efficient solutions [16]. Unlike TPP, fewer algorithms have been developed for PSP. Well-known algorithms developed for this phase include Earliest Finish Time (EFT) [17], which uses a greedy strategy to allocate processors, and Looking Ahead Sequencing Algorithm (LASA) [18], which has a stochastic structure to allocate processors based on a limited-lookahead rule. In the literature [19–22], most algorithms have used the described near-sight structures that significantly limit their capabilities. Therefore, the need to develop far-sighted algorithms is imminent. Such algorithms should be able to propose processors for allocation by considering the position of a task among the tasks of a level in a graph, which in the future will lead to the improvement of the objective functions. However, providing an intelligent method that monitors the status of the tasks in both phases and at the same time offers an effective solution is difficult. It should be kept in mind that monitoring requires features that must be carefully selected so that they have a complete representation of the problem space and tasks. These features should strike a proper balance between objectives that may conflict with each other (e.g., time and cost). Similarly, there is a trade-off between cost and energy consumption [23]. Therefore, it is important to determine the features that can help optimize the objectives of the problem.

The third aspect is the purposeful usage of the solutions generated by meta-heuristic algorithms. In the literature, many meta-heuristics have been developed for this group of NP-hard problems [20,24,25]. In each iteration of these meta-heuristics, useful information is produced. Although there are different selection operations in combination with different approaches, the operators use only part of the information and other useful information is wasted. This calls for methods that can provide high-quality results by better extracting knowledge from historical information.

Motivated by the need to provide an intelligent solution from the three aspects above in order to optimize energy consumption, time, and economic cost as the objectives of the two agents in IoT and fog-cloud computing (i.e., users and service providers), this paper studies the problem of scheduling and offloading multiple task graphs between IoT devices and fog-cloud. The main contributions of the paper are as follows:

- We design a novel automatic three-module algorithm for knowledge acquisition and resource allocation in PSP by using Genetic Algorithm (GA) and Random Forest (RF) classifier as the supervised machine learning techniques.
- We present a new GA algorithm by proposing a crossover operator to diversify the promising solutions in TPP for multiple IoT task graphs. Moreover, by defining a concept called bottleneck, a new mutator is developed in PSP to reduce the workload of MDCs/DCs by considering two conditions.
- We present and characterize the features for an IF-THEN rule format to place multiple IoT task graph applications in MDCs/DCs by an RF ensemble learning algorithm.
- We develop a method to refine the rules that play less role in improving the solution. In addition, a method to automatically adjust the maximum number of iterations based on the quality of the provided solutions without human intervention is proposed.
- We conduct intensive experiments to illustrate the accuracy of the proposed classifier and verify the efficiency of the proposed solution. The results demonstrate that the proposed algorithm outperforms other state-of-the-art algorithms.

The remainder of the paper is organized as follows: the next section reviews the related works. Section 3 describes the system model and problem formulations. The presented three-module method is explained in Section 4. Section 5 evaluates the performance of the presented method and compares it with the state-of-the-art algorithms. Section 6 summarizes the whole paper and suggests topics for future works.

2. Related works

This section briefly reviews some existing task scheduling algorithms applied in fog and cloud systems. The algorithms studied here are divided into three categories: meta-heuristic-based algorithms for multi-objective optimization, knowledge-based algorithms, and algorithms developed to improve the objective function in PSP. The end of this section provides a brief summary to the reviewed papers.

2.1. Meta-heuristic-based algorithms

In the literature, evolutionary optimization algorithms have been employed to find near-optimal solutions in fog-cloud computing environments to multi-objective task graph scheduling. Among nature-inspired algorithms, GA-based algorithms have been often used. The study in [26] proposed a Multi-Objective Genetic Algorithm (MOGA) to solve task graph scheduling in cloud environment. The optimization objective was to minimize makespan and energy consumption under the deadline constraint. They introduced a gap search algorithm as a neighborhood search to maximize the utilization of the resources. In [24], the authors investigated the task graph scheduling problem in a heterogeneous computing environment and developed a GA to optimize the energy consumption and execution time of the IoT devices. To maximize the number of tasks in parallel execution, they introduced a dynamic and lightweight pre-scheduling technique. The authors of [20] considered a three-layer system in which the cloud layer includes high-performance systems. Then, they developed a GA-based algorithm called Energy-efficient Makespan Cost-aware Scheduling (EMCS) that could efficiently balance the workload between the fog layer and the cloud layer to achieve good performance in terms of energy consumption, execution time, and cost. The study in [27] targeted the minimization of cost, makespan, and energy consumption under the given deadline and budget constraints by developing a hybrid Non-dominated Sorting Genetic Algorithm II (NSGA-II)-Owl Search Algorithm (OSA) algorithm. It introduced a hierarchical evolving method to ensure suitable exploration and exploitation and a chaotic operator as a local search method. Karimi et al. [21] developed an NSGA-II to reduce energy consumption and makespan. They introduced an intelligent semi-greedy algorithm to generate an efficient initial population and used the weighted sum method for PSP. Like the previous work, Xia et al. [22] also developed a heuristic for the initial population, showing that the initial solution effectively improves the two objective functions of makespan and energy consumption in their designed Adaptive Evolutionary Scheduling Algorithm (AESA). The approach in [28] uses a Particle Swarm Optimization (PSO) algorithm to propose a strategy for preventing premature convergence. The study, by using multiple swarms of different species of particles, could optimize four objectives of energy, makespan, cost, and load balancing for fog and cloud tiers. Khaledian et al. [19] developed an Improved Krill Herd (IKH) meta-heuristic algorithm, which optimized energy consumption, makespan, and cost. The meta-heuristic algorithm uses the dynamic frequency scaling search method in the initial population section to achieve fast convergence. To optimize the three objective functions of total tardiness, energy consumption, and cost, the authors of [29] employed a Multi-objective Salp Swarm Algorithm (MSSA) to explore the Pareto solutions and a local search method based on Iterative Greedy Algorithm (IGA) to refine the found solutions. In summary, in the presented papers, in addition to the techniques considered in the problem space, the changes applied in the specific operators of the algorithms or new neighborhood searches were introduced as contributions. However, in each iteration of these meta-heuristic algorithms, different solutions are produced, but it is not using past valuable experiences obtained in previous iterations, which is considered as a gap in the mentioned algorithms.

2.2. Knowledge-based algorithms

In order to benefit from past information, a learning schema by using the algorithm of learning automata was presented in [30] for task graph scheduling. The schema learns the optimal action through past experiences by a GA-based exploration method and repeats interactions with the environment. The work in [15] developed a three-step learning-based approach to scheduling IoT task graph applications. After exploration and recording the experiences in the first step, the second step uses a method to learn from the experiences and suggests solutions by interacting with the search space. The learning-based approach improves the solution in both TPP and PSP. To schedule IoT devices requests, the authors of [31] proposed a data mining-based algorithm. The algorithm employs several meta-heuristics and after the training phase and testing phase, it decides on which meta-heuristic to use to optimize the objectives in the scheduler. Using the combination of a variant of the subset sum problem and a k-means clustering technique, a two-phase task graph scheduling algorithm for dynamic resource provision in cloud environment is proposed in [32]. The study considered a centralized data recovery model for data transferring when a processing unit fails during a task execution. The evaluation results by considering the deadline constraint showed the effectiveness of the proposed algorithm over two other algorithms. In [33], Abbasi et al. improved Learning Classifier Systems (LCS) by introducing an intelligent Extended Classifier System (XCS) to find an optimal state for workload balancing in fog computing. They used a GA to search in the state set and a reinforcement algorithm for selecting the best state. In addition, they presented a classifier to avoid the random selection of actions by storing the sequence of the input conditions of the system. A two-stage approach to predict the task execution time was developed in [34]. The first stage uses an ensemble learning algorithm to learn information about the tasks and the environment, while the second stage predicts the final execution time. The study in [35] developed a two-stage scheduling approach called Parallel Reinforcement Learning Caledonian Crow (PRLCC) by considering the New Caledonian Crow Learning Algorithm (NCCLA)'s social and asocial learning behavior to create a global optimization algorithm. The capabilities of Q-learning algorithm in knowledge extraction and parallel computations in searching different directions of the problem space are also used. To reduce the task graph execution cost and execution time, Li et al. [10] presented a Knowledge-based Multi-Objective Estimation of Distribution Algorithm (KMOEDA) where four attributes of initial solutions, global search strategy, reliability-aware search strategy, and elite enhancement strategy showed its superior performance over other algorithms. The study in [11] suggested a Weighted Double Deep Q-Network-based Reinforcement Learning algorithm (WDDQN-RL) to minimize the makespan and cost. In order to improve the accuracy of the target value estimation in the WDDQN part, the authors introduced a dynamic coefficient-based adaptive balancing method. In addition, a dynamic sensing mechanism was presented for increasing the diversity of solutions. Dong et al. [12] introduced an actor-critic architecture to solve task graph scheduling achieving the makespan minimization. The architecture employed a Pointer network which consists of two Recurrent Neural Network (RNN) in TPP to extract the relevant information. The simulation experiment indicates the efficiency of the desired structure. However, most algorithms in this field have at least one of the following three limitations: (1) PSP was not considered, or even if it was considered, simple greedy-based heuristic algorithms like EFT were used; (2) no attention has been paid to refining the knowledge that prevents the algorithms from reaching optimal/near-optimal solutions; (3) the gained knowledge is only used in a limited time period, and is completely replaced by new knowledge in later periods.

2.3. *Algorithms to enhance PSP*

Some research papers suggest that approaches other than the greedy-based method of EFT [17] in PSP can show better performance. The authors in [18] presented a concept called Emphasized Processor (EP) in a heterogeneous distributed computing system to minimize the makespan of a task graph application. They state that although EFT is considered to be an effective method, there is no foresight in it. It was stated that, as the number of tasks of a level increases, EFT directs and focuses tasks on specific processors. In this way, they added the property EP to some tasks and added a processor ID to EP in TPP. This means that in PSP, that task does not follow EFT and is processed on a processor whose ID exists in EP. Although they showed that considering the EP property for some tasks leads to an improvement in makespan, for which task should EP be considered is an input parameter of the algorithm, thus a human expert is still needed. The work in [15] used the weighted sum method to select a suitable VM in cloud to assign a task to a VM in order to optimize the two objectives of energy consumption and makespan in PSP. After calculating the weighted sum for the available VMs, they are sorted in ascending order in a queue based on the result of the calculation. Then, it assigns a task to one of the three first VMs in the queue using a probability-based rule. They also used EP in their proposed method, but a learning-based method is used to determine which VM should a task be assigned to in PSP. An approach based on the A* search technique was presented in [36]. The proposed A*-based method demonstrated its efficiency in a competitive experiment. The authors of [37] suggested the Jordan Normal Form (JNF) for trainable parameter matrix following the Frobenius norm to Deep Kronecker Neural Network (DKNN) so that a hybrid JNF-DKNN algorithm could effectively monitor the available resources. They showed that monitoring can provide valuable insight that can dramatically improve performance. However, these efforts made to get rid of myopia in PSP have not led to the desired success in providing realistic far-sighted solutions.

2.4. *Motivation of this paper*

The literature review shows that many studies have been conducted in each aspect of the knowledge extraction, meta-heuristic-based methods development, and PSP, along with their limitations. In this paper, we consider all these aspects to present a holistic method. Our study provides a way to use the information produced in each iteration of a meta-heuristic as knowledge and improves decision-making in PSP for processor allocation by applying the knowledge. Furthermore, the developed algorithm can decide when to terminate by examining the generated solutions. Table 1 identifies the key elements of some most related works and compares them with our study.

3. System model and problem statement

In this section, we first explain the proposed architecture in the task scheduling process and then describe the task graph application model. We finally present a formal problem statement for the task scheduling problem. A list of key notations used in this paper along with their descriptions is given in Table 2.

3.1. *System architecture*

We consider an architecture with multiple IoT devices, multiple Fog Nodes (FNs), and multiple Cloud Nodes (CNs). The architecture is a distributed computing platform that executes large-scale offloaded IoT applications with collaboration between FNs and CNs. An overview of our system model is shown in Fig. 1.

Table 2
List of key notations used in the paper.

Notation	Description
N	Set of PCs
$N^{y,d}$	A PC in N, where y denotes the PC type ($y = 0$ if the PC is a FN, $y = 1$ if the PC is a CN), d is the index of the PC in the specified PC type
TG_n	A task graph with index n
V_n	Set of concurrent tasks of TG_n
$v_{n,i}$	The i^{th} task in TG_n
$t^{cp}_{v_{n,i}}$	Computation time of task $v_{n,i}$ on a PC
$q^{cp}_{v_{n,i}}$	Energy consumption required to compute task $v_{n,i}$ on a PC
$c^{cp}_{v_{n,i}}$	Computation cost of the task $v_{n,i}$ on a PC
$pred(v_{n,i})$	Set of immediate predecessors of task $v_{n,i}$
$AFT(v_{n,i})$	Actual finish time of task $v_{n,i}$ on a VM among all PCs
$EFT^{VM}_{v_{n,i}}$	Earliest finish time of task $v_{n,i}$ on a VM

Table 1
Comparative analysis of related works.

Reference	Application Properties		Architectural Properties		Solution Properties				
	Dependency Mode	Batch Placement	Fog Number	Cloud Number	Objective Function Number	Multi-objective Optimization Method	Knowledge Acquisition	Foresight in PSP	Automated Termination
[10]	Dependent	No	Not Seen	Single	Bi-Objective	Pareto Front	Yes	No	No
[11]	Dependent	Yes	Not Seen	Single	Bi-Objective	Pareto Front	Yes	No	No
[12]	Dependent	No	Not Seen	Single	Single objective	Weighted Sum	Yes	No	No
[15]	Dependent	No	Not Seen	Single	Bi-Objective	Pareto Front	Yes	Yes	No
[19]	Dependent	No	Multiple	Single	Multi-objective	Weighted Sum	No	No	No
[20]	Dependent	No	Multiple	Single	Multi-objective	Weighted Sum	No	No	No
[21]	Dependent	No	Multiple	Multiple	Bi-Objective	Pareto Front	No	No	No
[22]	Dependent	No	Not Seen	Single	Bi-Objective	Pareto Front	No	No	No
[24]	Dependent	Yes	Multiple	Multiple	Bi-Objective	Weighted Sum	No	No	No
[26]	Dependent	No	Not Seen	Single	Bi-Objective	Weighted Sum	No	No	No
[27]	Dependent	Yes	Not Seen	Multiple	Multi-objective	Pareto Front	No	No	No
[28]	Dependent	No	Multiple	Single	Multi-objective	Weighted Sum	No	No	No
[29]	Dependent	Yes	Not Seen	Multiple	Multi-objective	Pareto Front	No	No	No
[30]	Dependent	No	Not Seen	Single	Single objective	-	Yes	No	No
[31]	Independent	No	Multiple	Single	Multi-objective	Weighted Sum	Yes	No	No
[32]	Dependent	No	Not Seen	Single	Single objective	-	Yes	No	No
[33]	Independent	No	Multiple	Single	Single objective	-	Yes	No	No
[34]	Dependent	No	Not Seen	Single	Single objective	-	Yes	No	No
[35]	Independent	No	Not Seen	Single	Single objective	-	Yes	No	No
[37]	Dependent	No	Not Seen	Multiple	Multi-objective	Weighted Sum	Yes	Yes	No
Ours	Dependent	Yes	Multiple	Multiple	Multi-objective	Weighted Sum	Yes	Yes	Yes

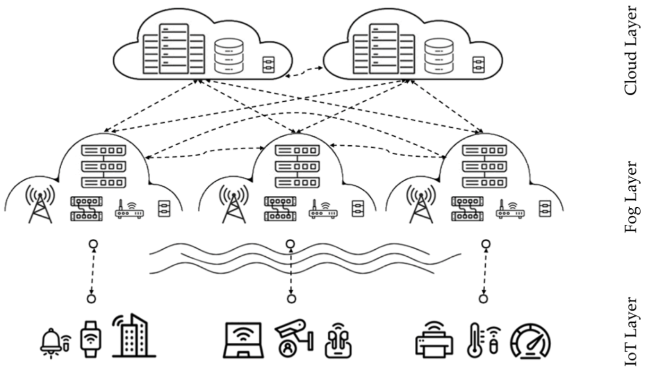


Fig. 1. Our system architecture.

The architecture has three layers in a hierarchical network. The bottom layer is the IoT layer that includes heterogeneous IoT devices such as wearable devices, smart home sensors, smartphones, tablets, thin-client, healthcare devices, etc. The middle layer represents the fog environment. It comprises of a set of FNs that consist of devices such as gateways, switches, and MDCs to provide services such as computing, storage, and networking. Each MDC includes homogeneous processor units (server processors or personal computers) in which there are several cores to process assigned requests. The cores can be controlled independently. Each FN includes a fog broker to handle the MDC utilization of the same FN. Also, fog brokers are close to the IoT devices to receive their requests and dispatch them among FNs and CNs in the network. Of course, if a fog broker corresponding to a FN sees that a request cannot be serviced in its own MDCs, it assigns that request to a suitable FN or CN. FNs in this layer are connected to the cloud layer to benefit from the advantages that CNs offer to improve service to IoT users. The upper layer represents the cloud computing layer, in which there is a pool of resources. In each CN, there are DCs and a cloud broker. Like fog brokers, cloud brokers are responsible for assigning requests to DCs or fog brokers. In fact, we can consider a broker (cloud broker or fog broker) as a scheduler and the duty of resource management is the responsibility of the broker. DCs include server processors as processor units. Both MDCs and DCs support DVFS. Since MDCs and DCs support virtualization technique, VMs are responsible to provide various services to the requests. In this study, we assumed that the number of VMs is equal to the number of cores of a processing unit and each VM can be assigned to one of the available cores in one time unit. So, brokers assign the requests to VMs in MDCs/DCs.

In this system, we consider each DC or MDC as a Processing Center (PC) and denote the set of PCs as $N = \{N^{y,d}\}$, where $y \in \{0, 1\}$ denotes the type of each PC, and d denotes the index of the PC in the specified PC type. Specifically, $y = 0$ if the PC is a FN and $y = 1$ if the PC is a CN. Thus, the set of PCs can also be represented as $N = \{N^{0,1}, N^{0,2}, \ldots, N^{0,nf}, N^{1,1}, N^{1,2}, \ldots, N^{1,nc}\}$, where nf denotes the number of FNs, and nc denotes the number of CNs. We assume that nodes of the fog layer are connected through a Local Area Network (LAN) and the connection of CNs with themselves and with FNs is through a Wide Area Network (WAN). In this case, the bandwidth of each link depends on which of the two PCs in N it is connecting. It is worth noting that link failures are not assumed in this study.

3.2. Task graph application model

Task graph applications are modeled in the form of DAGs. The nodes of a DAG are concurrent and have inter-dependencies. Since more than one task graph may be released by IoT devices at the IoT layer, we assign an index to each task graph. The application sent by the nth IoT device for processing is represented by a DAG $TG_n = (V_n, E_n)$, $\forall n \in \{1, 2, \ldots, ND\}$, where $V_n = \cup_{i=1}^{|V_n|} v_{n,i}$ denotes the set of concurrent tasks, $E_n = \{e_{n,i,j} | v_{n,i}, v_{n,j} \in V_n, i \neq j\}$ represents the set of directed edges between tasks, and ND denotes the number of IoT devices. Here, $e_{n,i,j}$ represents a precedence constraint between two tasks $v_{n,i}$ and $v_{n,j}$, where $v_{n,i}$ is the immediate predecessor of $v_{n,j}$, and $v_{n,j}$ is the immediate successor of $v_{n,i}$. A task $v_{n,i}$ may have more than one predecessor; in this case, $pred(v_{n,i}) = \{v_{n,i'} \in V_n | e_{n,i',i} \in E_n\}$ indicates the set of immediate predecessors of task $v_{n,i}$. Similarly, $succ(v_{n,i}) = \{v_{n,i'} \in V_n | e_{n,i,i'} \in E_n\}$ indicates the set of immediate successors of task $v_{n,i}$. We call a task $v_{n,i}$ the start task if $pred(v_{n,i}) = \varnothing$ and denote it as $v_{n,s}$ and $v_{n,i'}$ the end task if $succ(v_{n,i'}) = \varnothing$ and denote it as $v_{n,e}$. Each task $v_{n,i}$ has an amount of work that must be processed on the computing units which we denote as $v^w_{n,i}$. Besides, the non-negative weight of $e_{n,i,i'}$ that represents the data transferred from task $v_{n,i}$ to task $v_{n,i'}$ is denoted by $e^w_{n,i,i'}$. In TG_n, a task $v_{n,i}$ cannot be processed until the executions of all tasks in $pred(v_{n,i})$ have been completed and all associated data is transferred from $pred(v_{n,i})$ to $v_{n,i}$. We assume that preemptions are not allowed during the execution of a task and that the processing of a task continues from the time it starts until it is completed.

In a DAG TG_n, there may be more than one $v_{n,s}$ or $v_{n,e}$. In this case, we add two dummy tasks $v_{n,sd}$ and $v_{n,ed}$ to the graph as start task and end task, respectively, and consider $v^w_{n,sd} = 0$, $v^w_{n,ed} = 0$, $e^w_{n,sd,s} = 0$, $e^w_{n,ed,e} = 0$. We then redefine $v_{n,s} = v_{n,sd}$ and $v_{n,e} = v_{n,ed}$, so that the graph will have only one start task and one end task.

3.3. Problem statement

The problem of task graph scheduling in this study is to determine a mapping from the set of tasks $V_n \in TG_n$ to the set of PCs N in order to achieve certain optimization objectives. Different mappings will result in different sequences and orders of execution of the tasks, thus impacting the optimization objectives. In this paper, we study an optimization problem with the aim of simultaneously minimizing the makespan, cost, and energy consumption of MDCs/DCs. The following formulates each of these three objectives.

3.3.1. Makespan model

The goal of makespan optimization is to find a sequence of nodes in V_n for processing on N such that the execution time between the start time of $v_{n,s}$ and the completion time of $v_{n,e}$ is minimized. For this purpose, the total time between the Earliest Start Time (EST) of task $v_{n,s}$ and the Actual Finish Time (AFT) of task $v_{n,e}$ should be calculated.

In TG_n, the EST of a start task is equal to zero ($EST_{v_{n,s}} = 0$). The EST of a task $v_{n,i}$ on a VM ($EST^{vm}_{v_{n,i}}$) is when the execution of $pred(v_{n,i})$ is completed and all dependent data are transferred to the task, which is defined by Eq. (1).

$$EST^{vm}_{v_{n,i}} = \max_{v_{n,j} \in pred(v_{n,i})} \left\{ AFT(v_{n,j}) + t^{cm}_{e_{n,j,i}} \right\} \tag{1}$$

where $AFT(v_{n,j})$ is a function that returns the actual finish time of task $v_{n,j}$ among the VMs in all PCs and $t^{cm}_{e_{n,j,i}}$ is the communication time between the pair of dependent tasks $v_{n,j}$ and $v_{n,i}$, and it is calculated by Eq. (2).

$$t^{cm}_{e_{n,i,j}} = \begin{cases} 0 & \textit{if } v_{n,j} \textit{ and } v_{n,i} \textit{ are processed on the same MDC or DC} \\ \dfrac{e^w_{n,j,i}}{bw_{LAN}} & \textit{if } v_{n,j} \textit{ and } v_{n,i} \textit{ are both processed on MDCs} \\ \dfrac{e^w_{n,j,i}}{bw_{WAN}} & \textit{otherwise} \end{cases} \tag{2}$$

where bw_{LAN} and bw_{WAN} denote the bandwidths of LAN and WAN, respectively.

Moreover, the Actual Start Time (AST) of a task $v_{n,i}$ on a VM ($AST^{vm}_{v_{n,i}}$)

is calculated by Eq. (3).

$$AST^{vm}_{v_{n,i}} = \max\left\{avail_{vm},\ EST^{vm}_{v_{n,i}}\right\} \tag{3}$$

where $avail_{vm}$ is the time that vm has finished processing the previous assigned task on it and is ready to start the execution of task $v_{n,i}$. The Earliest Finish Time (EFT) of task $v_{n,i}$ on a VM ($EFT^{vm}_{v_{n,i}}$) is defined by Eq. (4).

$$EFT^{vm}_{v_{n,i}} = AST^{vm}_{v_{n,i}} + t^{cp}_{v_{n,i}} \tag{4}$$

where $t^{cp}_{v_{n,i}}$ is the computation time of task $v_{n,i}$ and is calculated based on Eq. (5).

$$t^{cp}_{v_{n,i}} = \begin{cases} \dfrac{v^{w}_{n,i}}{cpu_{N^{0,d}}} & \textit{if } v_{n,i} \textit{ is processed on an MDC} \\ \dfrac{v^{w}_{n,i}}{cpu_{N^{1,d}}} & \textit{if } v_{n,i} \textit{ is processed on a DC} \end{cases} \tag{5}$$

In Eq. (5), $cpu_{N^{0,d}}$ and $cpu_{N^{1,d}}$ denote the computation power of the MDCs and DCs, respectively. In this paper, we assume that DCs and MDCs have different computation power, and that the computation power of MDCs is less than that of the DCs.

Therefore, the overall execution time of TG_n is defined as Eq. (6):

$$T_n = AFT\left(v_{n,e}\right) \tag{6}$$

3.3.2. Cost model

Like the execution time model, the cost of processing a DAG TG_n also includes two parts: the cost of computing the tasks by the PCs and the cost of using the links to transfer the data between the PCs. The overall cost of processing TG_n is calculated by Eq. (7).

$$C_n = C^{cp}_{TG_n} + C^{cm}_{TG_n} \tag{7}$$

where $C^{cp}_{TG_n}$ is the cost of computing all the tasks of the task graph TG_n, and $C^{cm}_{TG_n}$ is the communication cost to transfer data from one PC to another using the links between the PCs.

The computation cost depends on the processing cost of each $N^{y,d} \in N$ and $v^{w}_{n,i}$. Therefore, $C^{cp}_{TG_n}$ is calculated by Eq. (8).

$$C^{cp}_{TG_n} = \sum\nolimits_{v_{n,i} \in V_n} c^{cp}_{v_{n,i}} \tag{8}$$

Here, $c^{cp}_{v_{n,i}}$ is the computation cost of the task $v_{n,i}$ and is calculated as Eq. (9).

$$c^{cp}_{v_{n,i}} = \begin{cases} v^{w}_{n,i} \times c_{N^{0,d}} & \textit{if } v_{n,i} \textit{ is processed on an MDC} \\ v^{w}_{n,i} \times c_{N^{1,d}} & \textit{if } v_{n,i} \textit{ is processed on a DC} \end{cases} \tag{9}$$

where $c_{N^{0,d}}$ and $c_{N^{1,d}}$ are the cost for computing each unit of work of $v_{n,i}$ on MDCs and DCs, respectively. In this paper, we assume that the costs of PCs in FNs and CNs are different.

The communication cost $C^{cm}_{TG_n}$ of the task graph is calculated by Eq. (10).

$$C^{cm}_{TG_n} = \sum\nolimits_{e_{n,i,j} \in E_n} c^{cm}_{e_{n,i,j}} \tag{10}$$

Here, $c^{cm}_{e_{n,i,j}}$ is the communication cost of $e_{n,i,j}$, and is calculated from Eq. (11) based on the corresponding link used to transfer the data.

$$c^{cm}_{e_{n,i,j}} = \begin{cases} e^{w}_{n,i,j} \times c_{LAN} & \textit{if } v_{n,i} \textit{ and } v_{n,j} \textit{ are both processed on MDCs} \\ e^{w}_{n,i,j} \times c_{WAN} & \textit{otherwise} \end{cases} \tag{11}$$

where c_{LAN} and c_{WAN} correspond to the communication cost of LAN and WAN, respectively.

3.3.3. Energy consumption model

The energy consumption of processing a DAG TG_n can be defined as the sum of the energy consumption when components involved in processing the DAG are active thus performing work (Q^{acv}) and the energy consumed when the PCs are idle (Q^{idl}), as depicted in Eq. (12).

$$Q_n = Q^{acv} + Q^{idl} \tag{12}$$

The amount of energy consumption Q^{acv} is defined as the sum of the energy consumed for computing the tasks ($Q^{cp}_{TG_n}$) and the energy consumed for transmitting the data for each pair of dependent tasks ($Q^{cm}_{TG_n}$), as depicted in Eq. (13).

$$Q^{acv} = Q^{cp}_{TG_n} + Q^{cm}_{TG_n} \tag{13}$$

The amount of energy consumption for computing the tasks is defined as Eq. (14).

$$Q^{cp}_{TG_n} = \sum\nolimits_{v_{n,i} \in V_n} q^{cp}_{v_{n,i}} \tag{14}$$

Here, $q^{cp}_{v_{n,i}}$ is the energy consumed to compute task $v_{n,i}$ and is calculated as Eq. (15).

$$q^{cp}_{v_{n,i}} = t^{cp}_{v_{n,i}} \times P_{acv} \tag{15}$$

where P_{acv} denotes the processing power of the PC when the PC is active.

The energy consumption due to data transmission between the tasks is defined as Eq. (16).

$$Q^{cm}_{TG_n} = \sum\nolimits_{e_{n,i,j} \in E_n} q^{cm}_{e_{n,i,j}} \tag{16}$$

Here, $q^{cm}_{e_{n,i,j}}$ is the energy consumed for data transmission between two dependent tasks $v_{n,i}$ and $v_{n,j}$, which is calculated as Eq. (17).

$$q^{cm}_{e_{n,i,j}} = \begin{cases} \dfrac{e^{w}_{n,i,j}}{bw_{LAN}} \times P_{LAN} & \textit{if } v_{n,i} \textit{ and } v_{n,j} \textit{ are both processed on MDCs} \\ \dfrac{e^{w}_{n,i,j}}{bw_{WAN}} \times P_{WAN} & \textit{otherwise} \end{cases} \tag{17}$$

where P_{LAN} and P_{WAN} are the transmission power of LAN and WAN, respectively.

The energy consumed by the PCs during idle time is calculated as Eq. (18).

$$Q^{idl} = \sum\nolimits_{N^{y,d} \in N} idle_{N^{y,d}} \times P_{idl} \tag{18}$$

where $idle_{N^{y,d}}$ is the amount of idling time of the PC $N^{y,d}$, and P_{idl} is the power of a PC during idle time. In this paper, we do not consider the idle power for transmission because it is typically insignificant.

3.3.4. Objective functions

The objective is to find a mapping from the tasks of each IoT task graph to the set of PCs N such that the three objectives of makespan (T), cost (C), and energy consumption (Q) are simultaneously minimized by a weighted sum. The objective function for each task graph TG_n is described by Eq. (19) below.

$$\min F(TG_n),\ \forall n \in \{1, 2, \ldots, ND\} \tag{19}$$

where

$$F(TG_n) = \mathscr{w}_1 \times T_n + \mathscr{w}_2 \times C_n + \mathscr{w}_3 \times Q_n \tag{20}$$

Here, T_n, C_n, and Q_n denote the makespan, cost, and energy consumption of the nth task graph. Besides, $\mathscr{w}_1$, $\mathscr{w}_2$, and $\mathscr{w}_3$ are the user-defined weighting parameters for the three objectives T_n, C_n, and Q_n, respectively.

4. Proposed method

In this section, we present a knowledge-driven method for scheduling multiple task graphs in the fog-cloud environment. We propose a Hybrid Genetic Algorithm-Random Forest (HGARF) algorithm, which is based on Genetic Algorithm (GA) and Random Forest (RF) to extract knowledge in the form of IF-THEN rules. To be specific, HGARF acquires accurate and functional decision rules from the solutions created by a meta-heuristic. The presented algorithm, which is considered as an expert system, extracts knowledge without having prior knowledge about the solution space and interacts with the environment by extracting rules and validating them in order to provide near-optimal solutions in this complex multi-objective problem. Therefore, GA is responsible for generating the solutions for the training set and RF is responsible for generating the rules from the solutions in the training set. The process of the proposed scheduler based on HGARF is illustrated in Fig. 2. After the task graphs are released by IoT devices, they are offloaded to HGARF for scheduling. The scheduler is embedded in the fog-cloud environment. HGARF then starts generating and deriving the rules. Thereafter, the extracted rules are used to find the most suitable solutions and present them to the related IoT devices.

The structure of HGARF comprises three modules: rule exploration, rule generation, and rule usage, where the second module is naturally executed after the results of the first module are obtained, and the third module is executed after the execution of the second module. This three-part sequence is repeatedly executed until the termination condition is met. In the following, we illustrate the implementation details for the three modules of HGARF.

4.1. Rule exploration

This section focuses on describing the GA-based rule exploration. Exploring the rules is an important process for obtaining effective solutions. During this process, the goal is to explore the environment well and to interpret rules for the environment in the training set. Since duplicate or similar solutions in the population may be produced by traditional GAs, in this paper, we pay special attention the diversity-preserving mechanism, which aims at generating a more diverse and unique set of solutions within a generation and across generations. Thus, the solutions provided by the proposed GA operators are unique, which helps to find the optimal/near-optimal rule faster. Such a diversity-preserving mechanism is applied to the two operators of crossover and mutation.

In the following, after introducing how to represent the solution, the proposed GA will be explained.

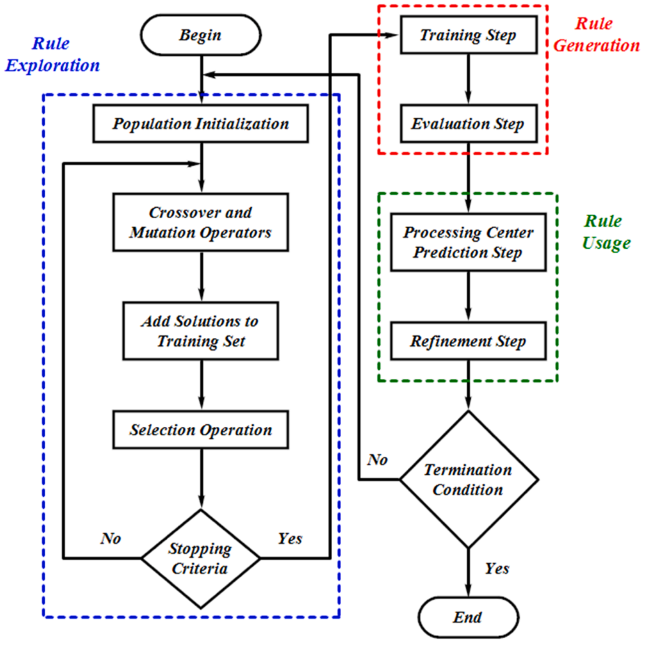


Fig. 2. Block diagram of the Hybrid GA-RF (HGARF) algorithm.

4.1.1. Solution representation

One of the important decisions made in the early stages of developing an algorithm is the representation of the solution. This leads to ease in understanding the schedule and its flexibility, especially in complex and large-scale problems. In the problem under study, the mapping of $V_n \in TG_n$ to N, the proper sequencing of tasks in TPP, and the recording of status of N in PSP are important topics that must be considered in the representation. In addition, the variables of the problem are numerical and discrete, and repeating tasks in each solution is not allowed. Hence, this problem is susceptible to the use of permutation-based encoding [38]. Besides, the need for producing cost-effective and high-quality schedules and for considering the nature of the problem in which there are two phases of TPP and PSP, strengthens the use of list-based scheduling [39] to decode the solutions. Thus, since the GA meta-heuristic is used, a two-dimensional chromosome is constructed for representation.

We use an example to provide a clear description of the solution representation. Suppose two DAGs TG_1 and TG_2 are generated by two IoT devices (i.e. $ND = 2$) as depicted in Fig. 3. In this case, a sample chromosome/individual in our representation is depicted in Fig. 4. As can be seen, the solution has two rows, and the number of columns is equal to the total number of tasks. In this representation, we call each column a gene, which represents a task and related information. The first row is called the task identifier (*tID*) and the second row the task information (*tInfo*). Each cell of the *tInfo* row contains a tuple in which there is some information related to the corresponding task.

Let $v_{n,j}$ be one of the predecessors of $v_{n,i}$, so the information placed in each tuple is listed as follows:

- *tInfoA*: the ID of the PC that processes the task $v_{n,i}$,
- *tInfoB*: the computation time of the task $v_{n,i}$ on the existing PC in *tInfoA* ($t^{cp}_{v_{n,i}}$),
- *tInfoC*: the computation cost of the task $v_{n,i}$ on the existing PC in *tInfoA* ($c^{cp}_{v_{n,j}}$),
- *tInfoD*: the energy consumption to compute the task $v_{n,i}$ on the existing PC in *tInfoA* ($q^{cp}_{v_{n,i}}$),
- *tInfoE*: the ID of the task $v_{n,j}$, where $v_{n,j} \in pred(v_{n,i})$ and its *AFT* is larger than *AFT* of all members of $pred(v_{n,i})$,
- *tInfoF*: $AFT(v_{n,j})$,
- *tInfoG*: the ID of the PC that processes the task $v_{n,j}$.

Note that in the TPP of rule exploration, the *tInfo* tuples are empty,

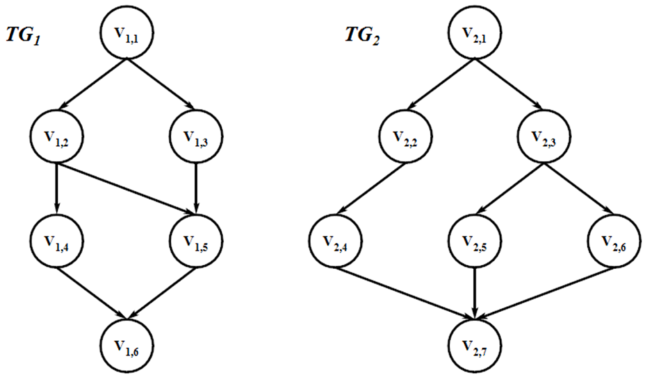


Fig. 3. An example of two DAG applications.

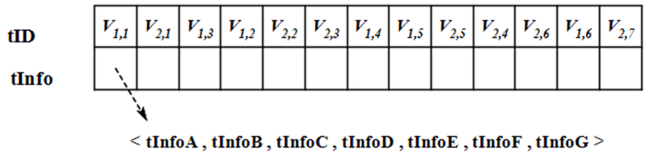


Fig. 4. A chromosome representing a sample solution.

and after assigning tasks to PCs in the PSP, this information will be filled in.

In order to traverse the DAGs to perform topological sort, the Breadth-First Search (BFS) algorithm is used. The traversal starts from $v_{n,s}$ and the tasks of each level of a DAG are placed in a group. Then, the groups are appended to the individual according to the level and the index of the graphs one by one. Since permutation leads to the emergence of a new solution, in the stated representation, permutation is performed in each group. For further explanation, consider the solution presented in Fig. 4 corresponding to the graphs of Fig. 3. The two tasks of $\left[[v_{1,1}],[v_{2,1}]\right]$ are placed in the same group and their group position in terms of level is one. Also, set $[v_{1,1}]$ is in the first position of group one and set $[v_{2,1}]$ is in the second position of group one. Similarly, group $\left[[v_{1,3},v_{1,2}],[v_{2,2},v_{2,3}]\right]$ is placed in the second position in terms of level, the set $[v_{1,3},v_{1,2}]$ which contains tasks from the same DAG is in the first position of the group, and the set $[v_{2,2},v_{2,3}]$ is in the second position of the group. Besides, task $v_{1,3}$ is in position one of the set and task $v_{1,2}$ is in the position two of the set.

4.1.2. Genetic algorithm

GA is one of the most widely used meta-heuristic algorithms that has been applied to various optimization problems, particularly task graph scheduling problems. GA is a stochastic search method that tries to produce better solutions by creating a collection of potential solutions (population) as initial solutions and evolving it in different generations through genetic operators (e.g., selection, crossover, and mutation) [40, 41]. In the population, solutions are usually referred to as individuals or chromosomes. Each individual is evaluated by its fitness, which is determined by the associated value of the objective function. The proposed GA consists of the following steps:

1. generating an initial population with *nPop* individuals.
2. applying the individuals to the environment and evaluating their fitness.
3. applying the crossover and mutation operators to generate new individuals.
4. applying the new individuals to the environment and evaluating their fitness.
5. appending the individuals to the training set.
6. applying the selection operator.
7. repeating steps 3 to 6 until the stopping criteria is met.

In this paper, we use GA for the exploration of the environment. This is because according to the results obtained from the combined compromise solution method in [42], GA is superior to the other well-known meta-heuristics such as Ant Colony Optimization algorithm (ACO) and PSO in terms of information collection. Details of our implementation of the proposed GA for the problem under study are given as follows.

4.1.2.1. Initial population. The first step of the optimization process with GA is the generation of the initial population. The initial population is consisting of *nPop* individuals. Each individual in this population represents a candidate and possible solution to the problem, which is also called the initial solution. The initial solutions are randomly and uniquely scattered in the solution space. Therefore, a powerful and unbiased search will be guaranteed in the developed GA. Note that *nPop* is a constant value and is one of the control parameters of GA. Also, the output of this step is the individuals whose row *tID* has tasks based on the representation stated in Section 4.1.1.

4.1.2.2. Fitness function. After determining the sequence of tasks, it is time to determine the fitness of an individual. This means, in PSP, the individual is assigned to *N* to determine its fitness value. During this process, the goodness of a solution according to the objective function can be calculated. Algorithm 1 demonstrates how the fitness of an individual is calculated.

An individual is assigned as an input value to Algorithm 1. Since the tasks of graphs are added to the individual in separate groups based on their levels, the groups of each level of graphs are selected in each iteration of the for-loop in line #1. Then, the selected groups are added to a container *C* in line #2. The container sorts the tasks in different selected groups based on the positions of the tasks. In this way tasks with position one of their groups are placed first, then tasks with position two, and so on. Consider Fig. 4 for example. In the first iteration of the for-loop in line #1, two tasks $v_{1,1}$ and $v_{2,1}$ are selected, thus $[v_{1,1},v_{2,1}]$ will be the content of container *C* after the execution of line #2. Accordingly, in the second iteration of the for-loop, the content of *C* after execution of line #2 will be: $\left[[v_{1,3},v_{1,2}],[v_{2,2},v_{2,3}]\right]$. The while-loop in line #3 is responsible for traversing the container *C* and assigning the tasks in the container to a suitable PC. Thus, in line #4, a task is selected from the beginning of *C* which has not been visited. In line #5, the value of *tInfo*. *tInfoA* is checked to see if it is empty or not. If it was empty, line #6 would be executed, otherwise line #11. The non-emptiness of *tInfo*. *tInfoA* occurs when the field *tInfoA* is set in some other parts of HGARF (for example, in the mutation operator). This means that this task will be processed without going through the process of finding the PC in line #6 and only by meeting the precedence constraints on the mentioned PC. The for-loop in line #6 ensures that all PCs are examined for task assignment, and the most appropriate center is allocated to the task according to the objective function. Note that in the examining process, it is assumed that the selected task is the final task in the current task graph (we name it the hypothetical final task). Thus, in line #8, Eq. (20) is utilized to evaluate the hypothetical final task in such a way that the assignment leads to the simultaneous minimization of all three objectives. The for-loop in line #13 also calculates the sum of the fitness of all the offloaded task graphs and returns it as the output of this algorithm through variable *retVal*.

Algorithm 1
Fitness function algorithm.

```
input : an individual;
output : summation of the best fitness value of all DAGs;
 1  For all groups related to the same level Do
 2      $C$ ← select and sort the tasks of the same level in the groups;
 3      While there is a task in $C$ Do
 4          select the first not-visited task $v_{n,i}$ in $C$;
 5          If $v_{n,i}.tInfo.tInfoA$ is empty Then
 6              For each $N^{y,d} \in N$ Do
 7                  calculate $EFT_{v_{n,i}}^{vm}$ value based on Equation (4);
 8                  assign task $v_{n,i}$ to $N^{y,d}$ which minimizes $F(v_{n,i})$;
 9                  fill the $tInfo$ tuple related to the task;
10          Else
11              assign the task to the PC mentioned in $tInfoA$;
12  $retVal \leftarrow 0$;
13  For each existing $TG_n$ Do
14      $retVal \leftarrow retVal + F(TG_n)$;
15  Return $retVal$;
```

4.1.2.3. Selection. Selection is a biased decision-making process whereby a new population is created based on the fitness values of the current population. This process is repeated in each generation of GA to keep the good individuals. In this paper, we used a roulette wheel method [43] in which two individuals are selected randomly and their fitness values are compared. Then, the individual with the better fitness value is transferred to the new population.

4.1.2.4. Crossover. The crossover operator is one of the main operators of GA, which is very effective in improving the performance of GA. This operator is used to search for new solutions (individuals) in the solution space. In this way, by combining two individuals (called parents) or making changes in an individual from the previous generation, new individuals (called offsprings) or a new individual are produced in the current generation. The process of producing new individuals is repeated in all generations of GA with the aim of producing better individuals.

In this study, according to the representation of the solution and the structure of the task graph, a new crossover technique named Horizontal Crossover Operator (HCO) is introduced. In order to create diversity in the solution, this operator puts the path of working on tasks of the same level on its agenda. HCO receives an individual as input and returns a new individual as output after applying fundamental changes to it.

The procedure of HCO is shown in Algorithm 2. This algorithm is repeated *nCros* times in each generation. To explain Algorithm 2, let us assume that the illustrated individual in Fig. 4 is the selected individual to apply HCO on. Therefore, we name the input individual as *newIndv*. Since the length of the longest level of the task graphs in Fig. 3 is four, *newIndv* has four levels. Suppose level 3 in *newIndv* is randomly selected in line #3. This level has two sets $[v_{1,4}, v_{1,5}]$ and $[v_{2,5}, v_{2,4}, v_{2,6}]$ where the position of the former is one and the latter is two. If position two is selected in line #4, then $selGrp = [v_{2,5}, v_{2,4}, v_{2,6}]$. To change the position of *selGrp* in *selLev* in line #5, we assume that the position of *selGrp* is changed to one and the position of the other set is changed to two in *selLev*. By executing line #6, the position of tasks in *selGrp* will change randomly. For example, $selGrp = [v_{2,4}, v_{2,6}, v_{2,5}]$ could be one potential output of this line. In this way, the contents of *selLev* could be equal to: $[[v_{2,4}, v_{2,6}, v_{2,5}], [v_{1,4}, v_{1,5}]]$. As illustrated in the example, the positions of the contents of level 3 in *newIndv* are changed and *newIndv* is passed to the fitness function algorithm as input in line #7 to calculate its fitness value. The variable *nRep* is an input value of this operator. The crossover operator is defined as $HCO(nRep)$, where *nRep* expresses the number of repetitions of lines #3 to #6. In this way, HCO is active in TPP and deals with diversity in solutions.

4.1.2.5. Mutation. The mutation operator is another operator of GA that plays a substantial role in the evolution of the solutions. Adding random-based new information to the solutions can lead the algorithm to escape from local optima. In the proposed GA, a new mutation operator is introduced, which is different from the traditional operators. This operator, called Processor-based Mutation Operator (PMO), makes changes to PSP in *tInfo* of tasks so that it can provide promising solutions in each generation by refining the previous solutions. In addition, a concept called *bottleneck* is introduced in PMO. The operator tries to improve the objective function in the new individual by following the bottleneck and considering it as a criterion for workload distribution in the set of PCs *N*.

Definition. Let assume that $\{N^{0,A}, N^{0,B}, N^{0,C}\} \in N$ and *wlA*, *wlB*, and *wlC* are the amount of assigned workloads to $N^{0,A}$, $N^{0,B}$, and $N^{0,C}$ respectively. If $wlA > wlB > wlC$, we say that $N^{0,A}$ is a *bottleneck* PC because the amount of workload assigned to it is greater than the others, and this can lead to the objective function being suboptimal.

In the problem under study, due to the fact that the applications are concurrent and consist of dependent tasks, and also because the greedy EFT algorithm is used in PSP, we adapt the idea from solutions such as EP [18] and PCs' ranking [15] to reduce the accumulation on a PC/PCs. In this study, a method to identify the bottleneck PC and a solution to transfer the load from one PC to another are proposed. Since in the rule exploration module of HGARF, creating diversity in solutions is on the agenda, the randomness of some decisions is injected into PMO in order to maintain the nature of mutation and to satisfy the goals of the rule exploration.

Algorithm 3 illustrates proposed PMO. It is invoked as $PMO(nRep)$, where *nRep* is as input parameter. The algorithm receives an individual at random and returns a new individual as a solution. It is repeated *nMut* times in each generation and appends its output to the training set. The PCs' Participation Percentage (PPP) in line #2 is a method to identify the bottleneck. Among the three objectives considered in this study, time is influential because the increase in processing time in a PC leads to a rise in energy consumption and cost and vice versa. Therefore, in the first step of PPP, the sum of the times that each PC spent on processing the assigned tasks in *newIndv* is calculated. Since PCs have different processing capabilities (i.e., number of cores and processing power), normalizing them in terms of processing capabilities is the second step in PPP. Thus, the amount of time each PC is engaged in processing is updated according to the processing capability. In line #3, PCs are maintained in container *C* based on normalized time values and are sorted in ascending order based on the time values. Therefore, the PC that has been involved in more processing than other PCs is placed in the last position of *C* with its time value higher than the others. Line #5 selects the task or tasks from the last PC in *C* and places it in *selT*. The selected task must meet two conditions. First, the task should be one of the tasks that are among the upper half of the levels of the DAG. Second, the out-degree of the task should be more than those of other tasks. For example, the task $v_{1,2}$ of TG_1 in Fig. 3 can be a candidate task that has met both conditions. These two conditions are considered so that the operator can be effective in transferring task/tasks from PCs that are assigned a lot of workloads to other PCs for faster convergence. In line #6 and #7, the field *tInfoA* is set in the task with a PC whose workload is low (here, PCs located in the initial positions of *C* are chosen randomly). Then, in line #8, the fitness value of *newIndv* is calculated. Note that the lines #5 to #7 in the for-loop of line #4 is repeated *nRep* times.

4.1.2.6. Stopping criteria. One of the most common stopping criteria in GA is the maximum number of generations. This study follows this practice and uses *nGen* as an input parameter for the maximum number of iterations. We point out that when the stopping criterion is met, the best solution observed in this module is stored in a container called *eliteSet*, which can be accessed jointly in two modules: rule exploration and rule usage. The role of this container in the module of rule usage will be discussed in Section 4.3.

Algorithm 2
Horizontal crossover operator.

```
input :  an individual from the current population at random, nRep;
output :  a new individual;
1   call the input individual as newIndv;
2   For 1 to nRep Do
3       select one of the levels of newIndv at random and call it selLev;
4       select one group in selLev at random and call it selGrp;
5       change the position of selGrp in selLev at random;
6       shuffle the genes' positions in selGrp;
7   calculate the fitness value of newIndv;
8   Return newIndv;
```

Algorithm 3
Processor-based mutation operator.

```
input : an individual from the current population at random, $nRep$;
output : a new individual;
call the input individual as $newIndv$;
calculate the PPP;
$C \leftarrow$ sort the members of $N$ based on PPP values in ascending order;
For 2 to $nRep$ Do
    $selT \leftarrow$ select task from the last PC in $C$ that satisfies two conditions;
    // condition 1: the task should be in the upper half of the DAG levels
    // condition 2: the outgoing edges of the task should be more than others
    set $selT.tInfoA$ to PC with the position of $rand[1,(0.5 \times C.length())]$;
    modify $newIndv$ with $selT$;
calculate the fitness value of $newIndv$;
Return $newIndv$;
```

4.2. Rule generation

RF is a well-known ensemble-learning method that uses bootstrap sampling and random feature subspace to generate and integrate multiple decision trees at training step [44]. RF with its flexible and easy-to-use structure can be applied to both classification and regression problems. Among the advantages of this algorithm, RF is robust against overfitting, has efficient performance on large data, and has quality methods for estimating missing data. In addition, there are effective mechanisms such as random oversampling and Synthetic Minority Oversampling TEchnique (SMOTE) algorithm to deal with unbalanced data sets [45]. All these advantages have led this algorithm to present good performance in different fields. Therefore, RF is used in the rule generation module so that the IF-THEN rules can be detached by tracking the path from the root node to the leaf node of these trees.

In RF, a collection of k classifiers $H = \{h_1, h_2, \ldots, h_k\}$ is constructed based on attribute selection measures such as gain ratio and gini index. Each classifier h_i is a decision tree constructed independently by a technique such as ID3, C4.5, or Classification and Regression Tree (CART) through bootstrap sampling and feature randomization. The classifiers of the collection H work in parallel on the given training set D. In $D = \{(O_1, G_1), \ldots, (O_n, G_n)\}$, there are n samples where a sample $s_i = (O_i, G_i)$ includes two parts: an order list O_i and a label $G_i \in G$. The order list is represented as $O_i = \left((X_1; Obj_1), (X_2; Obj_2), \ldots, (X_w; Obj_w)\right)$, where w indicates the number of tasks in an individual, $X_i \subseteq X$ indicates the set of features related to a task $v_{n,i}$, and Obj_i expressing the calculated objective function value related to the task $v_{n,i}$ when considered as a hypothetical final task. Note that, to calculate Obj_i, we use the three fields *tInfoB*, *tInfoC*, and *tInfoD* of the task $v_{n,i}$, as stated in the Fitness Function (Section 4.1.2.2). The feature vector $X = \left(X^1, X^2, \ldots, X^z\right)$ includes z features that are divided into two categories. The first category is related to the environment space and the second category is related to the solution space. The one feature of the environment space includes PC used time, energy, and cost (*uPc*), where if there are three PCs in N, then there will be three values in it as attributes. To calculate the three parameters of time, energy, and cost on a PC, we sum each of the three mentioned parameters for all tasks in a solution on the PC and categorize it based on what will be said about G. In the solution space category, there are four features: the position of a group in a level (*pGiL*), the position of a task in a set (*pTiS*), and the two fields of *tInfoE* and *tInfoG* that are extracted from *tInfo* which we rename as *infE* and *infG* here. Therefore, z is equal to five. We use the feature selection process presented in [46] to introduce the five mentioned features. Moreover, we consider a multi-class classification by the decision tree algorithm to classify the objective function values. To that end, given the smallest and largest objective function values of n samples in D, we determine three splitting points in the objective function values and consider $G = \left[G_1^c, G_2^c, G_3^c, G_4^c\right]$, where G_i^c is a class label with G_1^c being a category with the lowest value of the objective function.

The rule generation module consists of two steps. The first step is the training step in which the RF is constructed using the labeled samples of the training set. The rules will be generated in this step. In the second step, which is the evaluation step, the rules are applied to the test set. These two steps will be explained in the following in detail.

4.2.1. Training step

In this step, the k classifiers train in parallel on k different training subsets to generate the rules. The subsets are generated from the training set D using random sampling with replacement. To diversify the models created by decision trees, the decision tree classifiers use both C4.5 and CART algorithms, which are available in H with an equal number. The output of this step is the rules that are transferred to the next step. We call the rule generation procedure in this step rule generator, which is presented in Algorithm 4.

Algorithm 4 takes D and k as inputs and returns a rule set $R = \{(R_{1'}, G_{1'}), \ldots, (R_{n'}, G_{n'})\}$ as output, where the $R_{i'}'s$ represent the conditions and the $G_{i'}'s$ represent the conclusions. In the while-loop, the algorithm generates $w \times k$ decision trees, extracts some rules and appends the rules

Algorithm 4
Rule generator.

```
input : training set $D$, $k$;
output : rule set $R$;
$i \leftarrow 1$; $R \leftarrow \emptyset$;
While $i \le k$ Do
    $bagging_i \leftarrow \emptyset$; $subSet_i \leftarrow \emptyset$; $rule_i \leftarrow \emptyset$;
    $bagging_i \leftarrow$ Apply bootstrapSampling ($D$);
    Repeat
        $subList_i \leftarrow$ Apply featureSelection ($bagging_i$);
        treeC4.5← Apply constructC4.5 ($subList_i$);
        // or treeCART ← Apply constructCART (subListᵢ)
    Until the tree is made
    $rule_i \leftarrow$ Apply traverse (treeC4.5);
    // or ruleᵢ ← Apply traverse (treeCART)
    $R \leftarrow$ push_back ($rule_i$);
    $i \leftarrow i + 1$;
Return $R$;
```

to the rule set R. In line #4, the *bootstrapSampling()* method is invoked to prepare a class of samples D_i^s by bootstrap sampling. Note that for k iterations of while-loop, there will be $D^s = \{D_1^s, D_2^s, \ldots, D_k^s\}$ classes of samples where D_i^s is the class produced in ith iteration of the while-loop. During the sampling period in the ith iteration, the samples that were not selected to be placed in D_i^s are placed in D_i^{OOB}, where $D_i^{OOB} \in D^{OOB}$ is the Out-Of-Bag (OOB) dataset. In this way, D^{OOB} is built in parallel to D^s where $D^s \cap D^{OOB} = \varnothing$ and $D^s \cup D^{OOB} = D$. It is worth mentioning that D^{OOB} is used to obtain the classification accuracy after the training step. In this way, D is divided into two groups: the training set D^s and the testing set D^{OOB}. The repeat-until-loop is responsible for constructing the decision tree based on the current technique used. If the technique is C4.5, then gain ratio is used in each tree node's splitting process as the information-theoretic criterion, while if the technique is CART, then gini index is used. After constructing the tree, it is time to traverse the tree to extract the rules. In line #9, Depth-First Search (DFS) algorithm is used for traversing the tree to extract rules. In order to express the IF-THEN rules, we consider the format of $IF\langle condition\rangle THEN\langle conclusion\rangle$ as stated in [47], and develop the customized format for the problem under study. For instance, if $v_{n,i}$ and $v_{n,j}$ are two consecutive tasks in the same individual, the rule structure related to X_i and X_{i+1} can be expressed as follows:

$$\begin{aligned}&IF\langle \ldots, \big((pGiL = 2)\&\big(uPc = N^{0,1}\big)\&(pTiS = 3)\&\big(infG = N^{1,2}\big)\\&\quad \&\big(infE = v_{n,i'}\big); \mathscr{u}\big), \big(\big(uPc = N^{1,3}\big)\&(pGiL = 3)\&(pTiS = 1)\\&\quad \&\big(infE = v_{n,j'}\big)\&\big(infG = N^{1,1}\big); \mathscr{v}\big), \ldots\rangle THEN\langle G_2^c\rangle\end{aligned}$$

where the expressions on the right side of the "$=$" sign are attributes of the corresponding features (e.g., 2, $N^{0,1}$, 3), $\mathscr{u}$ and $\mathscr{v}$ represent Obj_i and Obj_{i+1}, respectively, $\langle condition\rangle$ represents $R_{i'}$ and $\langle conclusion\rangle$ stands for $G_{i'}$.

After extracting the rules, they are appended to the rule set R in line #10.

4.2.2. Evaluation step

Accuracy is one of the most appropriate metrics to measure the performance of RF. For this purpose, we use D^{OOB} and test each member of D^{OOB} by its corresponding trained tree. (e.g., D_k^{OOB} by the kth tree). This test is done to see if the tree can correctly predict the class label G_i of a test sample in $D_k^{OOB} \in D^{OOB}$ or not. For this purpose, Eq. (21) is used to obtain the accuracy.

$$acc = \frac{1}{|D^{OOB}|} \sum_{i=1}^{|D^{OOB}|} p\big(D_i^{OOB}\big) \tag{21}$$

where $|D^{OOB}|$ indicates the number of members in D^{OOB}, $p\big(D_i^{OOB}\big)$ gives the classified result of the ith sample in D^{OOB} by the rule sets, where the result is 1 if the estimate is correct and 0 otherwise.

After all test samples from a $D_k^{OOB} \in D^{OOB}$ have been examined, the prediction accuracy is calculated for all members of the sample test. In this study, if the accuracy is more than 50 %, the rules extracted from tree k remain in the rule set. Otherwise, the relevant rules will be deleted.

4.3. Rule usage

The knowledge obtained in the rule generation module should be used in the search space to check whether the knowledge can make a correct prediction to improve the objective function or not. For this purpose, the rule usage module performs two steps. The first step is to test the quality of the rules and this step is called processing center prediction. In this step a suitable PC is suggested to a task of an individual using the rules in rule set, so the resource will be allocated to the task. These suggestions are applied to all tasks. Afterwards, the tasks of the individual are assigned to N based on Algorithm 1 and without considering the rules. Then, the fitness function values obtained from the assignments using rules and without using rules are compared. After the first step, a second step is applied as a refinement step for the solutions proposed in the first step. The following discusses the details of the two steps.

4.3.1. Processing center prediction step

The procedure of the processing center prediction step is shown in Algorithm 5. This algorithm gets a population with *nPop* individuals, the rule set R, and the container of *eliteSet* as input values. Determining the sequence in the individuals of the population is random and unique. The container *eliteSet* is not empty before the execution of Algorithm 5, because this container, in the first iteration of HGARF, has kept the most suitable solution in the rule exploration module. If the rule usage module can find a better solution than that solution, it will replace the previous solution found in rule exploration module. Thus, at the end of the rule exploration and rule usage modules, there is only one solution in *eliteSet*, which is the best solution found until then in terms of the value of the objective function. In each iteration of HGARF, the individuals whose objective function values are in the class $G_1^c \in G$ are appended to this container. The for-loop in line #2 is repeated for all individuals of the input population. The responsibility of the for-loop in line #4 is to traverse the individual from position one to the end, where a not-visited task is selected in each iteration. Then, in line #5, the selected task is matched with the rules in R and the ID of PC $N^{y,d} \in N$ that leads to the minimum objective function value up to that moment is selected. The ID is inserted to the field *tInfoA* in line #6. After the described process in this for-loop is done for all tasks, the individual in line #7 is assigned to N so that $N^{y,d} \in N$ is allocated to the task based on the content of *tInfoA*. The individual selected by the for-loop in line #2 is assigned to N regardless of the rules in R in line #8. The values of the objective function obtained in line #7 and line #8 are compared and if the values belong to the same class of G_1^c or G_2^c, a positive score is considered, otherwise a negative score is considered. These scores are used to automatically adjust the termination condition of HGARF. Thus, line #15 checks if the sum of positive scores is more than the sum of negative scores, in which case Algorithm 5 returns *True* as the output. This means

Algorithm 5
Processing center prediction.

```
input :  a population $pop$, rule set $R$, a set of elites $eliteSet$;
output : a Boolean value $retVal$, $eliteSet$;
 1  $rewV \leftarrow 0$; $punV \leftarrow 0$; $retVal \leftarrow \emptyset$;
 2  For each individual $indv$ in $pop$ Do
 3      $newIndv \leftarrow indv$;
 4      For each task $t$ in $newIndv$ Do
 5          find the best $N^{y,d} \in N$ to assign $t$ to it according to $R$;
 6          set $tInfo$ of $t$ with the obtained $N^{y,d} \in N$ from $R$;
 7      assign $newIndv$ to $N$ and store its fitness value in $newIndvF$;
 8      assign $indv$ to $N$ and store its fitness value in $indvF$;
        // indvF is calculated by Algorithm 1
 9      If $indvF$ and $newIndvF$ belong to the same class of $G_1^c$ or $G_2^c$ Then
10          $rewV \leftarrow rewV + 1$;
11      Else
12          $punV \leftarrow punV + 1$;
13      If $newIndvF$ belongs to $G_1^c$ Then
14          $eliteSet \leftarrow$ push_back ($newIndvF$);
15  If ($rewV > punV$) ? ($retVal \leftarrow True$) : ($retVal \leftarrow False$);
16  If ($eliteSet.length() < (pop.length()/nGuarantee)$) Then
17      $retVal \leftarrow False$;
18  Return $retVal$, $eliteSet$;
```

that the repetition of HGARF should be reduced. In line #13, if the objective function value of the predicted solution using R belongs to the class G_1^c, that individual is appended to the container *eliteSet*. To explain the if-then-condition in line #16, let us assume that HGARF is in the first iteration of its execution. In this case, when starting the rule usage module, the container *eliteSet* will contain one individual. Let us also assume that solutions belonging to class G_2^c or G_3^c are produced every time the outer for-loop is repeated. In such a situation, *eliteSet* will remain with one individual and the positive score will increase to the point where it may lead to the return value *retVal* in this algorithm being *True*. Therefore, the execution of HGARF will end even if its solution may not be a competitive one. Further, even if there are a small number of solutions in *eliteSet*, these solutions may not provide the minimum fitness value. Therefore, line #16 is used to guarantee that the number of solutions available in *eliteSet* is not less than a threshold in hope of finding more competitive solutions.

4.3.2. Refinement step

The responsibility of this step is to refine the solutions in the container *eliteSet* from the output of Algorithm 5. In each iteration of Algorithm 5, many solutions may be appended to container *eliteSet*. However, not all of them will lead to the improvement of the objective function. Therefore, it is necessary to refine them and keep only one solution that provides the most optimal fitness value in the container. For this purpose, we apply the PMO mutator to each individual and compare the effects of the mutator on the individual before and after the PMO application. If the fitness value is improved after the execution of PMO, we will keep that solution. The process of applying PMO is done for all members of the container *eliteSet*. In the end, only one solution whose fitness value is better than the others is kept. It is worth mentioning that if the termination condition is satisfied (see Section 4.4), the solution in the container *eliteSet* is considered as the output of HGARF.

4.4. Termination condition

One important topic often discussed in search problems is when to end the search process to reach a solution. Maybe a suitable solution is found at the beginning of the search process and the algorithm cannot improve the solution until the end. Moreover, getting stuck in a local optimum is another issue often observed in these problems. This study considers the use of knowledge to overcome this issue by using past experiences and interactions with the stochastic environment. Therefore, this part tries to automatically adjust the termination condition of HGARF. To this end, it uses the output received from the first step (processing center prediction) of the third module (rule usage) and a simple calculation to decide whether the process of executing the three modules of HGARF should be repeated or not.

For this purpose, in the first round of execution of HGARF, unique and diverse solutions are generated, and the knowledge obtained in the processing center prediction step is used. If the condition stated in line #15 of Algorithm 5 is satisfied, this step receives a reward signal from the output of the first step in *retVal* ($= True$). Reward means that the algorithm believes that the provided solutions have a suitable quality according to the objective function and can end its search process. On the contrary, if the stated condition is not satisfied, this step receives the punishment signal through *retVal* ($= False$), which means continuing the search process. Therefore, there is an integer variable called *endV* in this step, whose value indicates the number of repetitions of HGARF. This variable is equal to zero at the beginning of the HGARF execution. Whenever a reward signal is received, one unit is subtracted from it, and if a punishment signal is received, three units are added to it. If the value inside *endV* is zero or less, the repetition of HGARF ends.

4.5. Time complexity

The time complexity of the proposed HGARF is analyzed as follows. The rule exploration module starts with the construction of a chromosome in which the BFS algorithm takes $O(ND \times L^2)$ time to visit all tasks of all DAGs, where ND is the number of incoming task graphs to the system and L is the maximum number of tasks for all task graphs. Algorithm 1 is executed for each chromosome. Therefore, the time complexity of Algorithm 1 is $O(L \times N)$, where N is the number of PCs in the systems. The time complexity of population initialization in the worst time is $O(np \times L \times N)$, where np is the size of *nPop*. Since chromosome construction and population initialization are only executed once, their time complexities become insignificant, and we focus on the time complexities of crossover, mutation, and selection operators for this module. Regarding the crossover operator, it is easy to see that the time complexity is $C = O(nc \times ((nrc \times m) + (L \times N)))$, where nc is the number of repetitions of Algorithm 2, nrc is the number of iterations of the for-loop is line #2, and m is the time to shuffle the gene's positions in line #6. The time complexity of mutation operator is $M = O(nm \times (N + N\log N + nrm + (L \times N)))$, where nm is the number of repetitions of Algorithm 3 and nrm is the number of iterations of the for-loop in Algorithm 3. In addition, we consider the time complexity of the number of generations and selection operator as g and np, respectively. Therefore, the overall time complexity of the first module is $F = O(g \times (C + M + np))$.

HGARF calls the rule generation module after the rule exploration module. The time complexity of RF depends on the number of decision trees in RF, the number of samples, and the number of features, which at most can be executed k, n, and z times, respectively. Hence, the time complexity of the second module in the worst case is $S = O(k \times z \times n\log n)$. In the third module, the time complexity of the processing center prediction step (Algorithm 5) is $P = O(np \times ((L \times z\log z) + (L \times N)))$. Moreover, the time complexity of the refinement step depends on the size *eliteSet* before applying this step which we name es and its complexity is $R = O(es \times (N + N\log N + nrm + (L \times N)))$. Therefore, the time complexity of the third module is $T = O(P + R)$. Finally, the number of repetitions of HGARF depends on *retVal*, which we call x. Hence, the overall time complexity of HGARF is $O(x \times (F + S + T))$.

Finally, we note that although in this work we considered three objectives (i.e., makespan, cost, and energy), it does not increase the asymptotic complexity of HGARF, as the three objectives are combined and evaluated as a single weighted sum. Thus, from the complexity's perspective, it is not much different from evaluating just a single objective.

5. Experimental results

This section presents the results of our experimentation to evaluate the performance of the proposed HGARF algorithm to solve the multiple task graph scheduling problem. Section 5.1 introduces the experimental settings, including the data set, characteristics of the problem space, the baseline algorithms, and the parameter configuration. Then, a set of performance metrics to measure the quality of the proposed method is provided in Section 5.2. Finally, Section 5.3 evaluates the algorithms using two statistical significance tests.

5.1. Experimental setup

5.1.1. Data set

The IoT task graph applications used in our experiments include four types of real-world graphs from the Pegasus workflow Generator [48]: CyberShake, Epigenomics, LIGO Inspiral, and SIPHT. These task graphs have various structures with flexible operations in services and low coupling specifications. Their graph structures are illustrated in Fig. 5. To create diversity in the size of the task graphs to analyze scalability, they are divided into small-size and large-size categories based on the

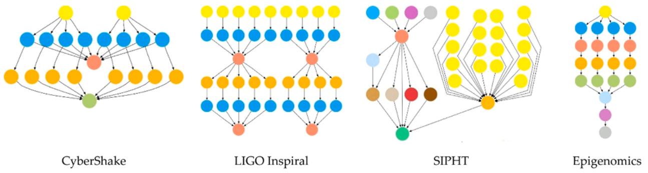


Fig. 5. The main structure of the used task graphs.

number of tasks. In the small-size category, the number of tasks is 100, while the number of tasks in the large-size is around 1000. With each execution of the workflow generator, different task graphs with various configurations in terms of task execution time and dependencies between them in each type/size are generated. In order to create diversity in the configuration of the task graphs, we ran the workflow generator five times for each type of graph (CyberShake, Epigenomics, LIGO Inspiral, and SIPHT). Therefore, the total number of task graphs for our simulation is $4 \times 2 \times 5 = 40$. Out of these graphs, 20 are small-size and 20 are large-size. To simulate the multiple task graphs submitted by IoT users, we randomly selected a configuration from each type of task graph and randomly put them together to build an application. Thus, each application includes four different types of task graphs.

5.1.2. Environment setting

The PCs in the considered environment have different processing capabilities which are represented by Million Instructions Per Second (MIPS). In our setting, the number of FNs and CNs are four and two, respectively. Two FNs have 4 cores and Two FNs have 6 cores, while the two CNs have 8 and 16 cores, respectively. Some details about the PCs settings are shown in Table 3. In this setup, four IoT devices are connected to four FNs, and each type of task graph is released by a specific IoT device and offloaded to the nearest FN. Besides, it is assuming the bandwidth of LAN and WAN are 4000 KB/s and 1000 KB/s, respectively. We also assume that the communication cost of LAN and WAN are 0.002 and 0.005 per data unit, respectively.

5.1.3. Baseline algorithms

In order to investigate the efficiency of the proposed HGARF, the results obtained by HGARF are compared with the following four algorithms under the same condition:

- Cooperative Multi-agent Offline Learning (CMOL) algorithm [15]: This is a pareto front-based multi-objective algorithm, and we added weighted sum to the different objectives. It also uses a learning-based method to reason about the environment. Its maximum number of iterations is 400.
- Multi-Objective Genetic Algorithm (MOGA) [26]: This is also an algorithm based on GA that uses a weighted sum method to calculate the fitness function. It uses a gap search algorithm to optimize resource utilization, and a tournament-based selection approach and a mutation operator for PSP. The maximum number of iterations of MOGA is 100.
- Energy-Efficient Makespan Cost-Aware Scheduling Algorithm (EMCS) [20]: This is a GA-based algorithm and uses the weighted sum method to solve the problem. It uses a parent selection strategy in the two-point crossover operator and a one-point mutation operator to avoid local optimum solutions. The maximum number of iterations of EMCS is 100.
- Only-GA (OGA): This is derived from HGARF but with RF removed. In order to improve its performance, we added two operators Block Search (BS) and Bad Selection Operator (BSO) [49] as neighborhood search and premature convergence prevention, respectively. In addition, we set the maximum number of iterations of the developed algorithm to 100.

Table 3
Characteristics of the processing centers in our setup.

	Number of Cores	Q^{idl} (W)	Q^{acv} (W)	Processing Power (MIPS)	Processing Cost (per time unit)
Fog Nodes	4	23	42	2500	0.02
	4	6	40	3000	0.03
	6	11	109	3500	0.04
	6	10	86	4000	0.05
Cloud Nodes	8	12	117	5500	0.08
	16	17	193	6000	0.09

5.1.4. Parameter configuration

Configuration of GA: Since there is no direct criterion for setting the parameters of meta-heuristic algorithms in scheduling problems and most approaches have used trial-and-error [50], we have employed a procedure to tune the parameters of our developed GA. For this purpose, we used the parameter values gained in the baseline algorithms to avoid spending time obtaining the values and have a fair comparison with those algorithms. The parameters were examined with respect to the baseline algorithms' values in the interval $[-60\ \%, 60\ \%]$ with a step size of 20 %, and the values that did not lead to a waste of time while improving the objective function were selected. As an example, Fig. 6 presents three separate typical turning results on the parameters of *nGen* and *nPop* on a large-size graphs. By comparing the performance, it is clear that setting $nGen = 50$ is sufficient for curve convergence; the fitness becomes stable and the ratio of fitness to time consumption decreases after that. The parameters of HGARF are set according to Table 4.

We set the three weight parameters w_1, w_2, and w_3 to 0.25, 0.25, and 0.5 respectively. This ensures that the user-oriented objectives (makespan and cost) share the same total weight (0.5) as the provider-oriented objective (energy consumption). In our experiments, all algorithms were implemented using Java with JDK 11 and the running environment of Intel(R) Core (TM) i7 CPU, 8 GB memory, and Windows OS.

Classification Accuracy of RF: To evaluate the classification accuracy of RF in HGARF, we compared the RF presented in HGARF, which we call RF-P, with RF-ID3, RF-C4.5, and RF-CART, which use ID3, C4.5, and CART as the tree construction techniques, respectively. For this purpose, the rule exploration module was executed only once and the data in the training set was used to construct the relevant decision trees. Here, the evaluation is based on different tree scales (i.e., number of decision trees in RF). To compare the accuracy of RF-P, RF-ID3, RF-C4.5, and RF-CART in terms of decision tree scales, we used all four types of graphs with large-size category. The experimental results are shown in Fig. 7.

As can be seen in Fig. 7, all four compared algorithms have low accuracy when the number of decision trees is 10. By increasing the number of decision trees from 10 to 20, the accuracy of the algorithms rapidly increases. Among the four task graphs, it is clear that RF-P offers the best accuracy in all the task graphs, while RF-ID3 has the worst accuracy. In addition, the lowest average accuracy for all algorithms is in LIGO Inspiral and the highest in Epigenomics. In terms of the difference in accuracy between the best and the second-best algorithms, the largest difference is 6.2 % (between RF-P and RF-CART in SIPHT), while the smallest difference is 1.6 % (between RF-P and RF-C4.5 in CyberShake). Overall, RF-P has shown better performance in terms of classification accuracy and thus could play an effective role in generating the rules.

Hereafter, we set the number of decision trees to be $k = 100$ in HGARF. According to Fig. 7, in CyberShake, SIPHT, and Epigenomics, the accuracy is about 0.9 when $k = 100$. Although the accuracy in LIGO Inspiral is less than 0.85 when $k = 100$, we use the same setting in order not to increase the run time in all task graphs.

5.1.5. Metrics

We considered the following four metrics to evaluate the

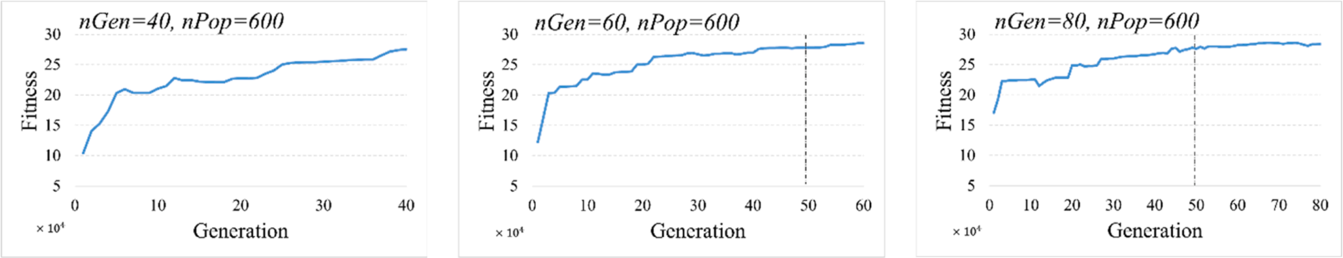


Fig. 6. Results of tuning the *nGen* parameter for large-size graphs.

Table 4
The parameters of HGARF algorithm.

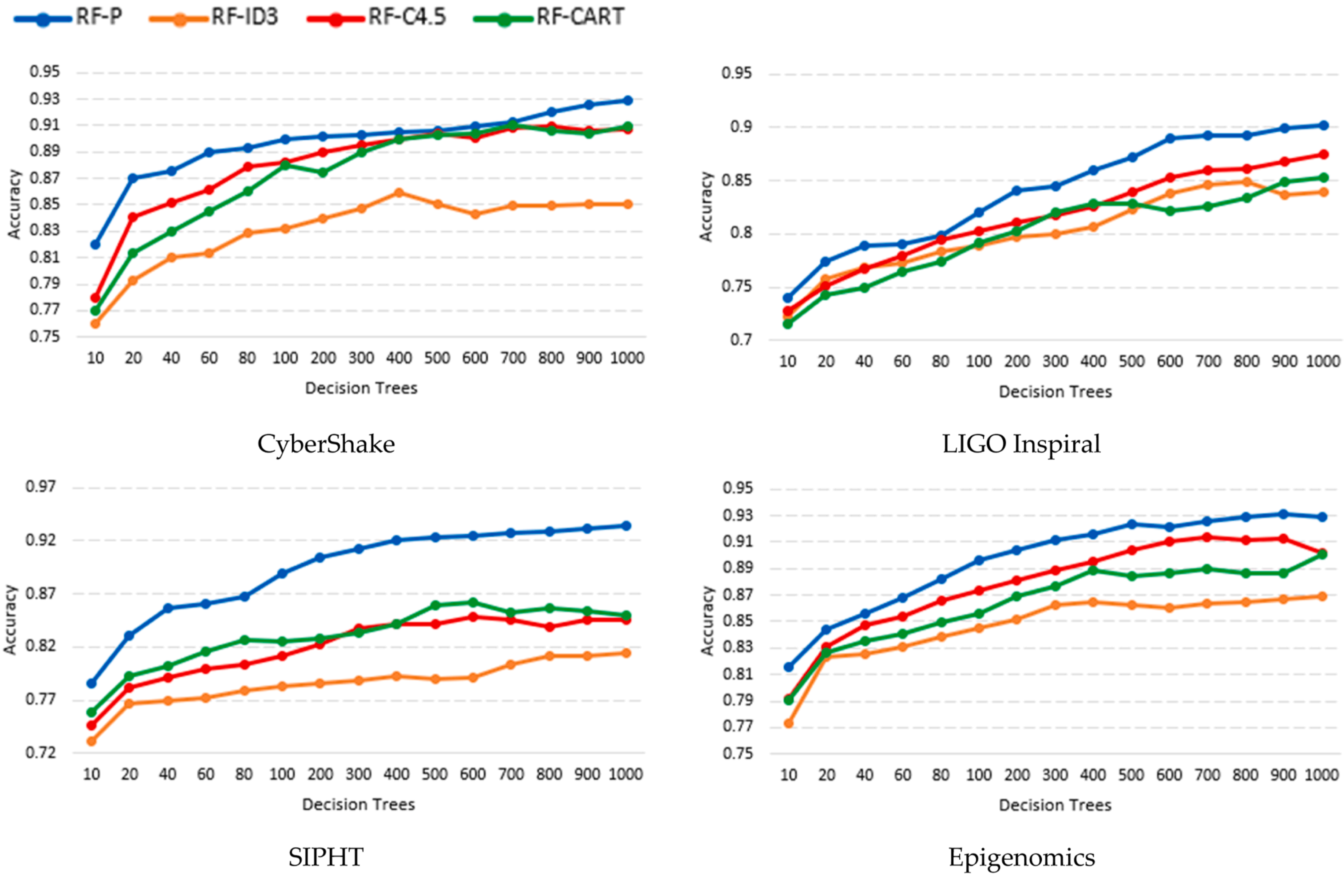
Parameter	Value
nPop	300 for small-size 600 for large-size
nCros	*nPop*
nMut	*nPop*
nGen	30 for small-size 50 for large-size
nRep	{2,4,8} for small-size {8,16,32} for large-size
nGuarantee	{5,6}

performance of HGARF.

- Objective Functions: The results of the five algorithms are evaluated separately based on the three objective functions: makespan, energy, and cost, as well as their weighted sum.
- Maximum Number of Iterations: To evaluate the impact of the maximum number of iterations on the performance of the algorithms, we multiply the maximum number of iterations of algorithms by increasing factors to check whether the quality of the obtained solutions also improves.
- Efficiency Ratios: This metric examines the efficiency of the algorithms by calculating the ratios of their performance and the lower bounds in terms of each of the three objective functions. These efficiency ratios provide measures on how far away each algorithm's performance is from that of the optimal solution. We use Schedule Length Ratio (SLR) [17], Energy Ratio (ER), and Cost Ratio (CR) for this purpose.
- Resource Utilization Rate: This metric measures the effectiveness and time spent by the PCs during the assigned tasks and refers to the maximum utilization of the capacity of the available PCs.

5.2. *Performance evaluation*

In this section, we compare HGARF with the four baseline algorithms



CyberShake

LIGO Inspiral

SIPHT

Epigenomics

Fig. 7. Classification accuracy of different RF techniques for different decision tree scales.

(CMOL, MOGA, EMCS, and OGA) and show their performance in terms of the objective functions, different iteration sizes, the efficiency ratio, resource utilization rate. For this purpose, several experiments are conducted to show the behaviors of these algorithms. In the following, each of these performance metrics is discussed separately.

5.2.1. *Comparison of objective functions*

For this metric, HGARF and CMOL, as learning-based algorithms, were each executed five times on the applications and the average results obtained from them were used. In contrast, MOGA, EMCS, and OGA, as non-learning-based algorithms, were each executed 10 times and the best results found were used. We did this to have a fair comparison considering that learning-based algorithms typically perform better than non-learning-based algorithms. In addition, we set the population size parameter of the three non-learning-based algorithms to be 400 and 600 for small-size and large-size applications, respectively.

Fig. 8 shows the makespan of the five algorithms on the two categories of graphs (small-size and large-size). The results show that the two learning-based algorithms perform better than the three non-learning-based algorithms in all task graphs for both small and large-size categories. In the class of learning-based algorithms, HGARF is superior in all instances. But, in the class of non-learning-based algorithms, the best is divided between the three. In particular, in small-size task graphs, OGA performs better than MOGA and EMCS in two instances, while each of EMCS and MOGA are better than others in one instance. In large-size task graphs, OGA is better than the two algorithms in two instances and similarly, MOGA performs better than other in two other instances. In the large-size task graphs, HGARF shows a significant improvement compared to the other four algorithms in CyberShake and Epigenomics. It is 16 % and 14 % better in CyberShake and Epigenomics, respectively, compared to CMOL, which is the second-best algorithm. The corresponding improvements in small-size task graphs are 12 % and 16 % in LIGO Inspiral and Epigenomics, respectively, compared to CMOL.

The results related to energy consumption are depicted in Fig. 9. In terms of energy, the learning-based algorithms are again superior to the non-learning-based algorithms. In addition, HGARF has also performed better in all instances. To compare in terms of the size of the task graphs, in the large-size category, a considerable distance between HGARF and CMOL (the second-best algorithm) is observed with HGARF providing 8 % and 9 % improvements in CyberShake and SIPHT, respectively.

The cost comparison of the five algorithms is shown in Fig. 10. The results once again show that HGARF outperforms the other algorithms in terms of this objective function and that CMOL, as another learning-based algorithm, has the second-best performance. In the small-size category, HGARF is better than CMOL with an improvement of 14 %, 9 %, and 15 %, respectively, in CyberShake, SIPHT, and Epigenomics. In the large-size category, HGARF exhibits a remarkable advantage over CMOL with an improvement of 21 % and 17 % in LIGO Inspiral and Epigenomics, respectively.

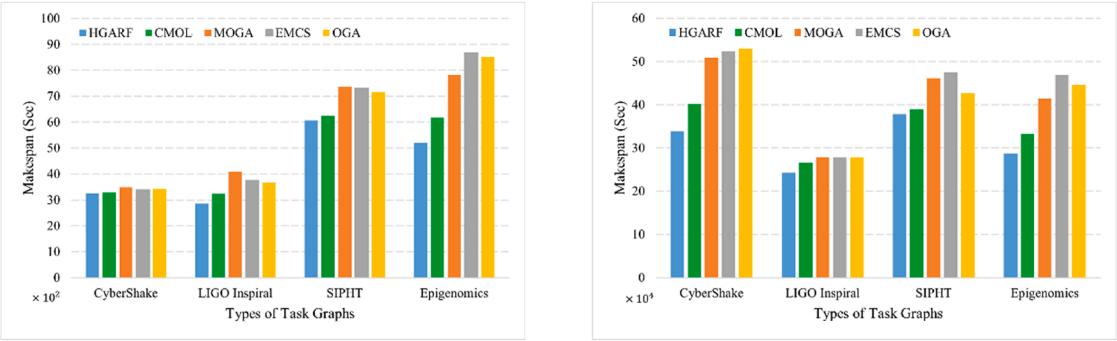
Finally, Fig. 11 compares the overall objective function of the five algorithms by the weighted sum of the three individual objectives using the weights as stated in Section 5.1.4. As expected, HGARF has the best performance for the weighted objectives due to its superiority for each individual objective. In particular, for small-size task graphs, HGARF outperforms CMOL, MOGA, EMCS, and OGA by 5.04 %, 13.24 %, 13.03 %, and 12.08 %, respectively, and for large-size task graphs, the respective improvements are 7.35 %, 18.91 %, 19.86 %, and 18.89 %. Overall, as the size of the task graphs increases, the superiority of HGARF over the other four algorithms also increases in terms of providing better-quality solutions.

Since HGARF automatically determines the maximum number of iterations during the execution, we recorded the maximum number of executions in this experiment, which are reported as follows: In the small-size task graph category, the maximum number of iterations is between 5 and 9, while in the large-size task graph category, the maximum number of iterations is between 7 and 16.

To conclude, the intelligent interaction of GA and RF with each other and with the problem space has led to HGARF's superior performance compared to the other four algorithms in all instances. Such intelligence stems from the following considerations. First, in HGARF, in addition to TPP, special attention has been paid to PSP by the introduction of the bottleneck concept and the presentation of the method in PMO to alleviate this issue. Second, the structure of a chromosome is designed in such a way that it can store valuable information from the environment in order to use them for knowledge extraction. Third, the mechanism presented in RF has helped HGARF to generate rules with high accuracy and to use the generated rules effectively. With these intelligent considerations, it is expected that HGARF would have a significant advantage over stochastic non-learning-based algorithms. Even compared to the state-of-the-art learning-based algorithm, its superiority can be anticipated from the methods presented in PMO and in the generation/usage of rules. As such, HGARF is able to significantly outperform the other algorithms as shown in Figs. 8, 9, 10, and 11.

5.2.2. *Comparison of maximum number of iterations*

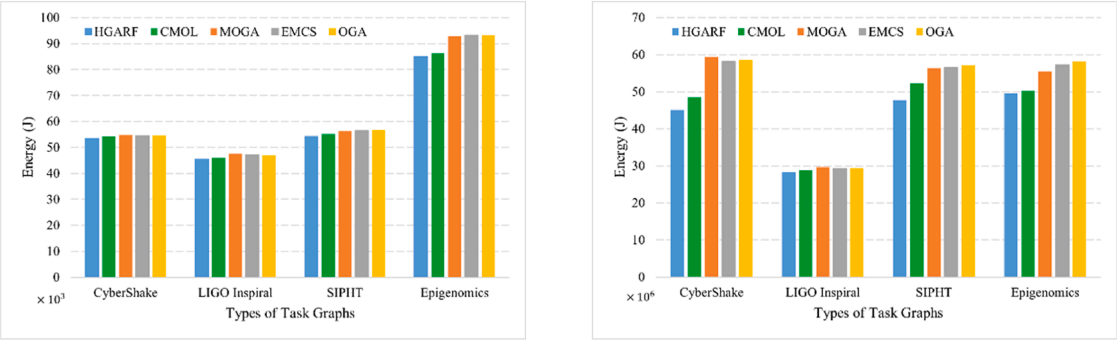
As mentioned in Section 4.4, HGARF automatically adjusts the maximum number of iterations. We expect that increasing the number of iterations will lead to the better extraction of quality rules, which can have a positive impact on the quality of the provided solutions. For this purpose, we use the maximum number of iterations recorded for the algorithm in Section 5.2.1 and omit the automation of the termination condition step in this experiment. As the recorded maximum number of iterations for small-size and large-size task graphs are in the intervals [5,9] and [7,16], respectively, we consider the average of the interval [5,9], which is 7, for small-size task graphs and the average of the interval [7,16], which is around 11, for large-size task graphs. As stated earlier, the maximum number of iterations of CMOL, MOGA, EMCS, and



small-size task graphs

large-size task graphs

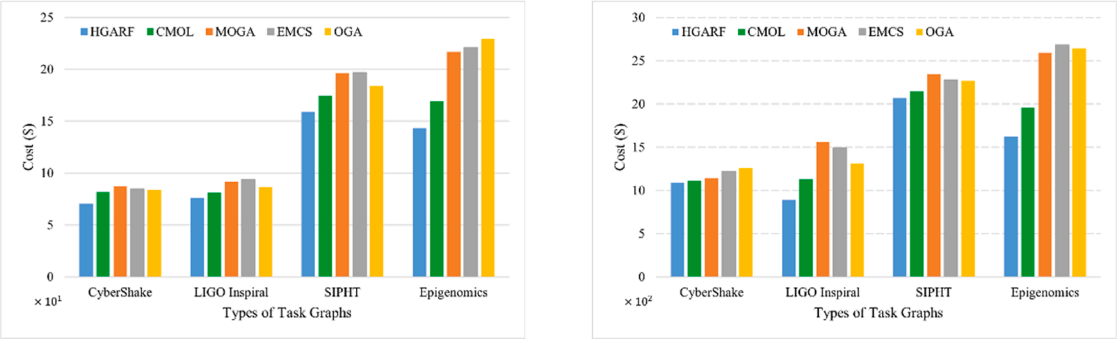
Fig. 8. Makespan comparison of the five algorithms for different task graphs.



small-size task graphs

large-size task graphs

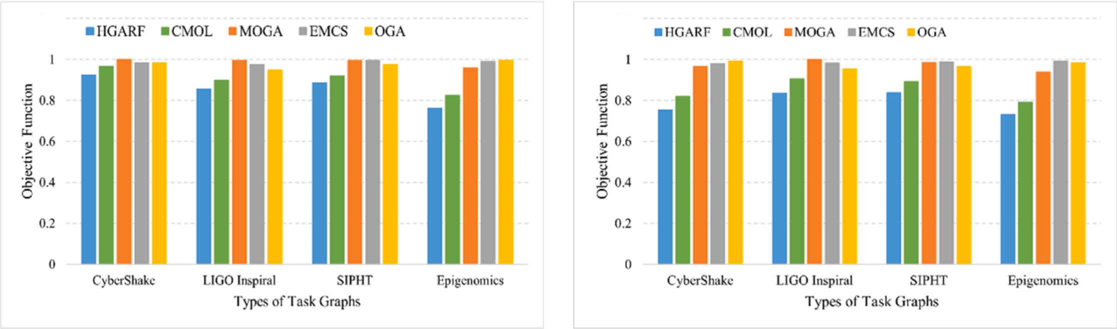
Fig. 9. Energy consumption comparison of the five algorithms for different task graphs.



small-size task graphs

large-size task graphs

Fig. 10. Average cost comparison of the five algorithms for different task graphs.



small-size task graphs

large-size task graphs

Fig. 11. Objective function comparison of the five algorithms for different task graphs.

OGA are 400, 100, 100, and 100, respectively. To this end, we increase the maximum number of iterations of the five algorithms by a factor of two and three, respectively, and compare the obtained results with those stated in Section 5.2.1. The percentage of improvements compared to the results of Fig.s 8, 9, and 10 are presented in Tables 5, 6, and 7, respectively, with the best results of each category marked in bold.

As seen in Table 5, there is a significant difference between the makespan results of learning-based algorithms and non-learning-based algorithms. Learning-based algorithms continue their convergence process by changing the increase factor from two to three. In contrast, the non-learning-based algorithms do not exhibit a specific pattern in convergence and improvement of results. It is observed that, in some examples, an increase factor of three even leads to a decrease of performance compared to an increase factor of two. Moreover, all improvements are less than 10 % from the previous results for both large-size and small-size categories. On the other hand, learning-based algorithms by changing the factor from two to three presented a considerable progress on convergence, although the speed of convergence decreased for large task graphs compared to small graphs. Comparing the two algorithms in the learning-based class, the convergence in HGARF is clearly better. Specifically, with an increase factor of two, the average improvements by HGARF for small-size and large-size task graphs are 27.07 % and 20.79 %, respectively, while they are 18.80 % and 11.34 % for CMOL. With an increase factor of three, this superiority trend continues for HGARF: the average improvements by HGARF for small-size and large-size task graphs are 39.37 % and 28.72 %, respectively, while they are 25.66 % and 14.18 % for CMOL.

As with the makespan, the performance of non-learning-based

Table 5
The percentage of improvement of makespan of the five algorithms with the increase of the maximum number of iterations.

Increase factor	Algorithms	Small-size Task Graphs				Large-size Task Graphs			
		Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
× 2	HGARF	**20.81 %**	**29.13 %**	**25.78 %**	**32.55 %**	**27.67 %**	**18.29 %**	**20.78 %**	**16.41 %**
	CMOL	17.11 %	20.69 %	15.36 %	22.05 %	13.04 %	11.50 %	8.28 %	12.54 %
	MOGA	7.40 %	6.24 %	3.19 %	6.01 %	2.04 %	3.11 %	1.83 %	1.14 %
	EMCS	6.44 %	5.03 %	1.98 %	4.67 %	1.86 %	0.28 %	0.62 %	1.54 %
	OGA	6.37 %	4.66 %	4.73 %	5.28 %	1.73 %	0.60 %	1.78 %	4.51 %
× 3	HGARF	**33.04 %**	**47.28 %**	**36.08 %**	**41.10 %**	**29.71 %**	**32.25 %**	**30.84 %**	**22.09 %**
	CMOL	19.64 %	28.65 %	23.93 %	30.42 %	18.05 %	15.70 %	9.62 %	13.34 %
	MOGA	3.02 %	5.52 %	2.19 %	7.52 %	2.67 %	2.38 %	1.72 %	4.10 %
	EMCS	4.44 %	3.27 %	2.93 %	4.81 %	1.35 %	0.37 %	0.52 %	1.19 %
	OGA	5.39 %	6.25 %	4.34 %	6.48 %	2.65 %	1.10 %	2.31 %	3.49 %

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

Table 6
The percentage of improvement of energy consumption of the five algorithms with the increase of the maximum number of iterations.

Increase factor	Algorithms	Small-size Task Graphs				Large-size Task Graphs			
		Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
× 2	HGARF	**26.06 %**	**30.75 %**	**29.12 %**	**31.50 %**	**33.80 %**	**29.72 %**	**30.21 %**	**26.17 %**
	CMOL	21.23 %	18.74 %	14.59 %	18.36 %	13.01 %	15.26 %	9.48 %	14.39 %
	MOGA	0.45 %	3.20 %	1.63 %	3.81 %	1.52 %	2.34 %	-	0.69 %
	EMCS	2.62 %	5.07 %	3.06 %	3.53 %	0.87 %	0.28 %	-	0.05 %
	OGA	2.70 %	8.26 %	3.19 %	4.25 %	2.07 %	3.72 %	-	0.11 %
× 3	HGARF	**34.59 %**	**43.13 %**	**35.64 %**	**42.91 %**	**28.34 %**	**31.55 %**	**31.62 %**	**25.44 %**
	CMOL	18.93 %	21.84 %	24.77 %	26.02 %	16.36 %	15.09 %	14.58 %	17.33 %
	MOGA	2.88 %	5.68 %	4.09 %	5.38 %	3.28 %	0.42 %	-	3.06 %
	EMCS	4.16 %	4.09 %	3.61 %	4.11 %	1.03 %	0.58 %	-	-
	OGA	5.46 %	4.31 %	5.28 %	5.67 %	1.96 %	-	1.67 %	2.08 %

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

Table 7
The percentage of improvement of cost of the five algorithms with the increase of the maximum number of iterations.

Increase factor	Algorithms	Small-size Task Graphs				Large-size Task Graphs			
		Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
× 2	HGARF	**19.05 %**	**27.60 %**	**26.84 %**	**32.17 %**	**28.07 %**	**20.06 %**	**19.32 %**	**18.66 %**
	CMOL	15.99 %	19.74 %	16.02 %	24.29 %	13.98 %	12.73 %	10.71 %	11.29 %
	MOGA	7.58 %	5.32 %	5.18 %	6.74 %	1.53 %	4.55 %	2.09 %	1.02 %
	EMCS	4.16 %	2.79 %	3.52 %	2.07 %	1.82 %	2.01 %	1.69 %	1.05 %
	OGA	5.32 %	5.68 %	6.59 %	7.04 %	1.29 %	1.58 %	4.80 %	2.03 %
× 3	HGARF	**32.06 %**	**50.20 %**	**38.73 %**	**40.37 %**	**31.46 %**	**33.93 %**	**32.17 %**	**22.56 %**
	CMOL	17.52 %	30.95 %	24.70 %	33.28 %	19.28 %	17.79 %	10.54 %	12.06 %
	MOGA	4.58 %	5.03 %	6.17 %	6.37 %	1.01 %	2.27 %	6.68 %	1.92 %
	EMCS	5.19 %	3.24 %	4.99 %	3.34 %	1.51 %	1.06 %	1.85 %	0.99 %
	OGA	4.82 %	7.56 %	7.29 %	5.81 %	2.43 %	1.79 %	3.66 %	3.82 %

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

algorithms compared to learning-based algorithms in terms of the energy consumption is significantly poor, as shown in Table 6. In fact, the non-learning-based algorithms even show no improvement in seven instances (indicated by the “-“ sign in the table). Moreover, the maximum improvement achieved by the three algorithms in this class is merely 8 %. In contrast, the learning-based algorithms demonstrate much better performance. Notably, the highest average improvement achieved by HGARF is 39.07 %, occurred in small-size graphs and the lowest average improvement is 29.24 % in large-size graphs, both with an increase factor of three. As for CMOL, the highest average improvement occurred in the small-size graphs is 22.89 % with an increase factor of three, while the lowest average improvement is in the large-size category is 13.04 % with an increase factor of two. These results again highlight the superiority of HGARF over CMOL.

The superiority of HGARF over the other four algorithms continues in Table 7 for the cost. As can be seen, HGARF has led to more than 50 % improvement in one instance, while CMOL does not get close to HGARF in terms of the cost improvement. MOGA, EMCS, and OGA again show very poor performance with maximum improvement less than 8 % for all instances. Moreover, these three algorithms do not show a clear pattern in the improvement, as it is expected that with an increase in the number of iterations, their improvements will also be better, which is not the case.

In summary, we can conclude that, for the two learning-based algorithms HGARF and CMOL, the more time they are given for exploration, the more favorable results they will get. In addition, the size of the problem space negatively affects their convergence speed. For the three non-learning-based algorithms MOGA, EMCS, and OGA, due to their stochastic nature, they cannot necessarily increase the speed of convergence by increasing the number of iterations. Finally, regarding the superiority of HGARF over CMOL, we can see that HGARF leads to about 40 % better performance than CMOL on average, mainly due to HGARF’s capability for: (1) paying special attention to PSP by using the PMO algorithm; (2) providing a high-accuracy method for classification and rule generation; and (3) refining the rules with less effect on the solution in the rule usage module.

5.2.3. Comparison of efficiency ratios

SLR is a metric that has been utilized to assess the makespan in various algorithms. It normalizes the makespan of an algorithm by a lower bound (LB) to report the efficiency of the algorithm. Since four task graphs with different structures and attributes were utilized in our study, the makespan LB of each task graph is calculated using the critical path of the task graph and then used to normalize an algorithm's makespan. The critical path is the longest path between $v_{n,s}$ and $v_{n,e}$ that is processed on a PC with the highest processing capability. The SLR metric is calculated by Eq. (22).

$$SLR = \frac{makespan}{LB(makespan)} \tag{22}$$

The denominator of the fraction in Eq. (22) calculates the LB on makespan, and since the actual makespan must be greater than the LB, the SLR will not be less than one. Therefore, an algorithm is better if its SLR is closer to one. Table 8 presents the comparison of the SLR metric derived from their average executions amongst the HGARF, CMOL, MOGA, EMCS, and OGA algorithms according to Fig. 8, where the best results are marked in bold.

From Table 8, we can see that HGARF substantially outperforms the other algorithms on the basis of the SLR metric. In particular, within the class of learning-based algorithms, HGARF performs better than CMOL, and the class of non-learning-based algorithms clearly has weaker performance. HGARF has an SLR of less than 1.4 for small-size task graphs, which shows that its makespan for such graphs is very close to the optimal. For large-size task graphs, HGARF's SLR is less than 4 for all the graphs, and the difference of it from the other three algorithms is much greater than that of small-size task graphs.

To evaluate the efficiency of the algorithms in terms of energy consumption and cost, the algorithms were executed on a PC with the lowest energy consumption and cost, respectively, which allows us to calculate the minimum energy consumption and the minimum cost for each task graph, and we are using them as the energy and cost lower bounds. In this way, we propose two metrics called ER and Cost Ratio CR, similarly to the SLR, and they are defined in Eqs. (23) and (24).

$$ER = \frac{energy}{LB(energy)} \tag{23}$$

$$CR = \frac{cost}{LB(cost)} \tag{24}$$

Again, the closer these ratios are to one, the better the algorithm is in terms of the energy/cost efficiency. Table 9 and Table 10 present the ERs and CRs of the four algorithms for different task graphs. The results again confirm the superiority of HGARF compared to the other algorithms for these two metrics. In particular, HGARF's ratios are less than 1.5 for small-size task graphs and less than 5 for large-size task graphs. Such superiority is again due to HGARF's use of experiences to generate knowledge such that over time the algorithm learns which sequences and assignments bring it closer to the goal. Overall, the SLR, ER, and CR, metrics confirm the effectiveness of the proposed HGARF algorithm.

5.2.4. Resource utilization rate

The Resource Utilization Rate (RUR) metric measures the percentage of VMs in PCs used by the applications in the fog-cloud environment. It indicates how much of the PCs' capacity is being utilized over time. It is calculated by Eq. (25).

$$RUR = \frac{VM\ busy\ time}{VM\ total\ available\ time} \times 100\% \tag{25}$$

For this metric, each algorithm was executed five times on both small-size and large-size applications, and the average results obtained from them are shown in Fig. 12. The results indicate that HGARF has the best RUR compared to other algorithms, which is more than 93 % for small-size task graphs. For both sizes, HGARF's rate is not less than 85 %, while the highest rate recorded by other algorithms is around 85 %. This superiority can be attributed to the good performance of HGARF in PSP, in particular the development of the PMO mutator, which is effective in properly utilizing the resources by identifying the bottleneck PCs and considering the two conditions. Another reason is the use of rules that can lead to intelligent allocation of resources.

5.3. Evaluations based on statistical analysis

This section compares the performance of our algorithm and that of the baseline algorithms based on two statistical significance tests, namely, the Friedman test and the Wilcoxon signed-rank test.

5.3.1. Friedman test

Friedman test is a non-parametric test used to rank and compare the developed algorithm with the baseline algorithms. This test examines whether the mean ranks among the algorithms are the same at the significance level of 0.05. A comparison of the performance of HGARF with the performance of CMOL, MOGA, EMCS, and OGA based on the values of the objective functions is presented in Table 11. Considering that the P-value is less than 0.05, the null hypothesis (i.e., there is no difference between the performance of the algorithms) is rejected and it can be concluded that the algorithms do not have the same performance. HGARF is ranked first with an average value of 1.00, thus having the best performance. Among the baseline algorithms, CMOL performs better than others.

5.3.2. Wilcoxon signed-rank test

The Wilcoxon signed-rank test is a nonparametric statistical test used to determine whether two related samples have the same population mean ranks. We use it to verify the statistical differences in the performance of HGARF with CMOL, MOGA, EMCS, and OGA at the significance level of 0.05. The results of this test for pairs of algorithms based on the objective function values are shown in Table 12. Since the significance value is smaller than 0.05, it indicates a significant difference between HGARF and the other algorithms and that HGARF performs better than those algorithms. Also, the table shows the pairwise comparison among other algorithms as well. We can see that there is no statistically significant difference between MOGA and EMCS, MOGA and OGA, as well as EMCS and OGA, since the P-values of these three comparisons are greater than 0.05.

6. Conclusion and future directions

This paper investigated the problem of multiple task graph scheduling in heterogeneous distributed computing systems in which multiple IoT devices, multiple fog nodes, and multiple cloud nodes are available.

Table 8
SLR comparison of the five algorithms for different task graphs.

Algorithms	Small-size Task Graphs				Large-size Task Graphs			
	Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
HGARF	**1.23**	**1.16**	**1.29**	**1.37**	**3.25**	**3.45**	**3.41**	**3.84**
CMOL	1.25	1.31	1.32	1.62	3.83	3.79	3.57	4.45
MOGA	1.32	1.66	1.60	2.06	4.86	3.96	4.21	5.53
EMCS	1.28	1.52	1.58	2.28	5.00	3.95	4.34	6.27
OGA	1.30	1.49	1.51	2.24	5.06	3.95	3.89	5.96

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

Table 9
ER comparison of the five algorithms for different task graphs.

Algorithms	Small-size Task Graphs				Large-size Task Graphs			
	Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
HGARF	**1.37**	**1.29**	**1.42**	**1.34**	**2.98**	**2.84**	**3.36**	**3.47**
CMOL	1.43	1.48	1.55	1.49	3.62	3.65	4.03	4.78
MOGA	1.56	1.51	1.62	1.88	4.92	4.89	5.62	6.04
EMCS	1.55	1.50	1.59	1.71	4.68	4.51	5.64	5.27
OGA	1.55	1.63	1.60	1.72	4.71	4.52	4.99	5.36

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

Table 10
CR comparison of the five algorithms for different task graphs.

Algorithms	Small-size Task Graphs				Large-size Task Graphs			
	Cyber.	LIGO.	SIPHT	Epi.	Cyber.	LIGO.	SIPHT	Epi.
HGARF	**1.14**	**1.33**	**1.37**	**1.13**	**4.31**	**2.99**	**4.97**	**3.57**
CMOL	1.32	1.41	1.50	1.34	4.43	3.79	5.26	4.31
MOGA	1.41	1.60	1.69	1.71	4.52	5.26	5.74	5.71
EMCS	1.37	1.63	1.70	1.74	4.84	5.03	5.59	5.91
OGA	1.36	1.50	1.58	1.81	4.99	4.42	5.56	5.82

Note: Cyber. = CyberShake; LIGO. = LIGO Inspiral; Epi. = Epigenomics.

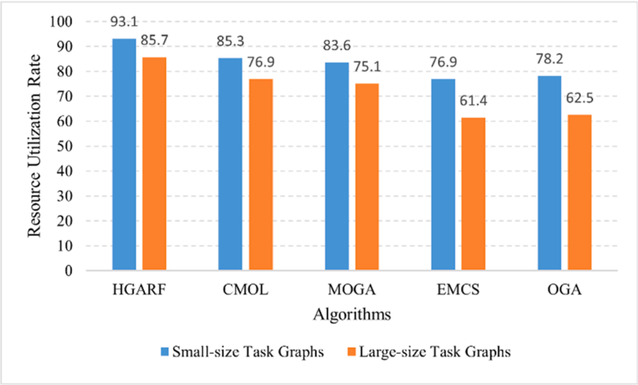


Fig. 12. Resource utilization rate comparison of the five algorithms for different task graphs.

Table 11
Results of Friedman test based on objective function values.

Algorithm	Mean rank	Statistic	P-value
HGARF	1.00		
CMOL	2.00		
MOGA	4.00	26.500	0.000025
EMCS	4.38		
OGA	3.63		

Table 12
Results of Wilcoxon signed test based on objective function values.

Algorithm pair	Difference of Medians	Wilcoxon statistic	P-value
HGARF-CMOL	−0.05902	2.521	0.012
HGARF-MOGA	−0.15346	2.521	0.012
HGARF- EMCS	−0.14951	2.521	0.012
HGARF-OGA	−0.14332	2.521	0.012
CMOL-MOGA	−0.09444	2.521	0.012
CMOL-EMCS	−0.09049	2.521	0.012
CMOL-OGA	-0.0843	2.521	0.012
MOGA-EMCS	0.003956	0.420	0.674
MOGA-OGA	0.010143	0.420	0.674
EMCS- OGA	0.006187	1.680	0.093

The goal is to simultaneously minimize the three objectives of makespan, energy consumption, and cost through a weighted sum. To solve the problem, a three-module intelligence algorithm was developed, which explores the problem space and derives IF-THEN rules by using a Genetic Algorithm (GA) and a Random Forest (RF) classifier, respectively. The designed algorithm, which we call Hybrid GA-RF (HGARF), can interact with the environment to achieve quality rules, and automatically terminate its activity when it reaches a suitable convergence. In the GA part of the developed HGARF, a method was presented in the crossover operator to create diversity in a chromosome for problems with multiple task graphs. In addition, by defining a concept called bottleneck, a mutator was introduced to reduce the workload based on multiple fog/cloud nodes. In the RF part, in addition to introducing features to construct the decision trees, a format for extracting and recording rules was introduced. To show the superiority of the proposed HGARF, its performance was compared with that of four other state-of-the-art algorithms on four task graphs with small and large sizes. The results show that the developed algorithm was able to provide better performance than all other four algorithms in all testing instances. In addition, the accuracy of the developed RF was compared with that of three other techniques, and favorable results were also observed.

Considering the complexity of the studied problem, a method in the refinement step of the third module of HGARF was introduced to remove the rules that play less role in convergence from the rule set. Although this method is effective by taking advantage of the introduced mutator, designing new mutation methods that can maintain valuable rules in each task graph by segmenting the levels of each task graph can be a path for the future development of HGARF. In addition, since the rules in the rule set cannot completely cover the problem space due to the limitation in the maximum number of iterations, embedding a fuzzy IF-THEN rule in the processing center prediction step to make decisions in uncertainty conditions can increase the efficiency of our algorithm. Real-time tasks are one of the workload types that the fog-cloud environment hosts for processing. Although more rules are added to HGARF’s training set to increase the quality of its decision-making, in the presence of real-time tasks at the early stages, it may not be able to have the necessary quality for task placement. Therefore, including a semi-greedy heuristic in HGARF is another direction that we will consider in the future. Lastly, applying HGARF to optimize the execution of other IoT applications such as bag-of-tasks and using heterogeneous ensemble learning classifiers in the second module can be other paths for future research.

CRediT authorship contribution statement

Hadi Gholami: Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Hongyang Sun:** Writing – review & editing, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] W.A. Kassab, K.A. Darabkh, A–Z survey of Internet of Things: Architectures, protocols, applications, recent advances, future directions and recommendations, J. Netw. Comput. Appl. 163 (2020) 102663, https://doi.org/10.1016/j.jnca.2020.102663.

[2] X. Ma, et al., Real-time multiple-workflow scheduling in cloud environments, IEEE Trans. Netw. Serv. Manage. 18 (4) (2021) 4002–4018, https://doi.org/10.1109/TNSM.2021.3125395.

[3] X. Ma, et al., Real-time virtual machine scheduling in industry IoT network: A reinforcement learning method, IEEe Trans. Industr. Inform. 19 (2) (2022) 2129–2139, https://doi.org/10.1109/TII.2022.3211622.

[4] H. Gao, et al., Com-DDPG: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles, IEEe Trans. Veh. Technol. (2023), https://doi.org/10.1109/TVT.2023.3309321.

[5] P. Li, et al., EPtask: Deep reinforcement learning based energy-efficient and priority-aware task scheduling for dynamic vehicular edge computing, IEEE Trans. Intell. Veh. (2023), https://doi.org/10.1109/TIV.2023.3321679.

[6] D. Tychalas, H. Karatza, A scheduling algorithm for a fog computing system with bag-of-tasks jobs: simulation and performance evaluation, Simul. Model. Pract. Theory. 98 (2020) 101982, https://doi.org/10.1016/j.simpat.2019.101982.

[7] M. Lin, et al., Scheduling co-design for reliability and energy in cyber-physical systems, IEEE Trans. Emerg. Top. Comput. 1 (2) (2013) 353–365, https://doi.org/10.1109/TETC.2013.2274042.

[8] L. Benini, A. Bogliolo, G. De Micheli, A survey of design techniques for system-level dynamic power management, IEEe Trans. Very. Large Scale Integr. VLSI. Syst. 8 (3) (2000) 299–316, https://doi.org/10.1109/92.845896.

[9] H. Gholami, H. Sun, Toward automated algorithm configuration for distributed hybrid flow shop scheduling with multiprocessor tasks, Knowl. Based. Syst. (2023) 110309, https://doi.org/10.1016/j.knosys.2023.110309.

[10] M. Li, D. Pi, S. Qin, Knowledge-based multi-objective estimation of distribution algorithm for solving reliability constrained cloud workflow scheduling, Cluster. Comput. (2024) 1–19, https://doi.org/10.1007/s10586-023-04022-w.

[11] H. Li, et al., Weighted double deep Q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud, Cluster. Comput. (2022) 1–18, https://doi.org/10.1007/s10586-021-03454-6.

[12] T. Dong, et al., Workflow scheduling based on deep reinforcement learning in the cloud environment, J. Ambient. Intell. Humaniz. Comput. (2021) 1–13, https://doi.org/10.1007/s12652-020-02884-1.

[13] J. Wang, et al., Hybrid physics-based and data-driven models for smart manufacturing: Modelling, simulation, and explainability, J. Manuf. Syst. 63 (2022) 381–391, https://doi.org/10.1016/j.jmsy.2022.04.004.

[14] L. Perotin, H. Sun, P. Raghavan, Multi-resource list scheduling of moldable parallel jobs under precedence constraints, in: Proceedings of the 50th International Conference on Parallel Processing, 2021, https://doi.org/10.1145/3472456.3472487.

[15] H. Gholami, M.T. Rezvan, A cooperative multi-agent offline learning algorithm to scheduling IoT workflows in the cloud computing environment, Concurr. Computat.: Pract. Exp. 34 (22) (2022) e7148, https://doi.org/10.1002/cpe.7148.

[16] Y. Kumar, S. Kaul, Y.-C. Hu, Machine learning for energy-resource allocation, workflow scheduling and live migration in cloud computing: State-of-the-art survey, Sustain. Comput.: Inf. Syst. 36 (2022) 100780, https://doi.org/10.1016/j.suscom.2022.100780.

[17] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parall. Distrib. Syst. 13 (3) (2002) 260–274, https://doi.org/10.1109/71.993206.

[18] H. Gholami, R. Zakerian, A list-based heuristic algorithm for static task scheduling in heterogeneous distributed computing systems, in: 2020 6th international conference on web research (ICWR), IEEE, 2020, https://doi.org/10.1109/ICWR49608.2020.9122306.

[19] N. Khaledian, et al., IKH-EFT: An improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment, Sustain. Comput.: Inf. Syst. 37 (2023) 100834, https://doi.org/10.1016/j.suscom.2022.100834.

[20] R. Sing, et al., EMCS: an energy-efficient makespan cost-aware scheduling algorithm using evolutionary learning approach for cloud-fog-based IoT applications, Sustainability 14 (22) (2022) 15096, https://doi.org/10.3390/su142215096.

[21] S. Karami, S. Azizi, F. Ahmadizar, A bi-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization, Appl. Soft. Comput. 151 (2024) 111142, https://doi.org/10.1016/j.asoc.2023.111142.

[22] Y. Xia, et al., Dynamic variable analysis guided adaptive evolutionary multi-objective scheduling for large-scale workflows in cloud computing, Swarm. Evol. Comput. 90 (2024) 101654, https://doi.org/10.1016/j.swevo.2024.101654.

[23] N. Abbas, et al., Joint computing, communication and cost-aware task offloading in D2D-enabled Het-MEC, Comput. Netw. 209 (2022) 108900, https://doi.org/10.1016/j.comnet.2022.108900.

[24] M. Goudarzi, et al., An application placement technique for concurrent IoT applications in edge and fog computing environments, IEEe Trans. Mob. Comput. 20 (4) (2020) 1298–1311, https://doi.org/10.1109/TMC.2020.2967041.

[25] M.I. Khaleel, A dynamic weight–assignment load balancing approach for workflow scheduling in edge-cloud computing using ameliorated moth flame and rock hyrax optimization algorithms, Fut. Generat. Computer Syst. 155 (2024) 465–485, https://doi.org/10.1016/j.future.2024.02.025.

[26] A. Rehman, et al., Multi-objective approach of energy efficient workflow scheduling in cloud environments, Concurr. Comput.: Pract. Exper. 31 (8) (2019) e4949, https://doi.org/10.1002/cpe.4949.

[27] H. Li, et al., Chaotic-nondominated-sorting owl search algorithm for energy-aware multi-workflow scheduling in hybrid clouds, IEEe Trans. Sustain. Comput. 7 (3) (2022) 595–608, https://doi.org/10.1109/TSUSC.2022.3144357.

[28] D. Subramoney, C.N. Nyirenda, Multi-Swarm PSO Algorithm for Static Workflow Scheduling in Cloud-Fog Environments, IEEe Access 10 (2022) 117199–117214, https://doi.org/10.1109/ACCESS.2022.3220239.

[29] Z. Sun, et al., Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud, Expert. Syst. Appl. 228 (2023) 120401, https://doi.org/10.1016/j.eswa.2023.120401.

[30] H. Izadkhah, Learning based genetic algorithm for task graph scheduling, Appl. Comput. Intell. Soft Comput. 2019 (2019), https://doi.org/10.1155/2019/6543957.

[31] D. Rahbari, Analyzing meta-heuristic algorithms for task scheduling in a fog-based IoT application, Algorithms 15 (11) (2022) 397, https://doi.org/10.3390/a15110397.

[32] V. Singh, I. Gupta, P.K. Jana, A novel cost-efficient approach for deadline-constrained workflow scheduling by dynamic provisioning of resources, Fut. Gener. Comput. Syst. 79 (2018) 95–110, https://doi.org/10.1016/j.future.2017.09.054.

[33] M. Abbasi, et al., Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems, Comput. Commun. 153 (2020) 217–228, https://doi.org/10.1016/j.comcom.2020.02.017.

[34] T.-P. Pham, J.J. Durillo, T. Fahringer, Predicting workflow task execution time in the cloud using a two-stage machine learning approach, IEEE Trans. Cloud Comput. 8 (1) (2017) 256–268, https://doi.org/10.1109/TCC.2017.2732344.

[35] B.M.H. Zade, N. Mansouri, M.M. Javidi, A two-stage scheduler based on new Caledonian crow learning algorithm and reinforcement learning strategy for cloud environment, J. Netw. Comput. Appl. 202 (2022) 103385, https://doi.org/10.1016/j.jnca.2022.103385.

[36] Y.-K. Kwok, I. Ahmad, On multiprocessor task scheduling using efficient state space search approaches, J. Parallel. Distrib. Comput. 65 (12) (2005) 1515–1532, https://doi.org/10.1016/j.jpdc.2005.05.028.

[37] P.V. Reddy, K.G. Reddy, An energy efficient RL based workflow scheduling in cloud computing, Expert. Syst. Appl. 234 (2023) 121038, https://doi.org/10.1016/j.eswa.2023.121038.

[38] H. Sun, W.-J. Hsu, Y. Cao, Competitive online adaptive scheduling for sets of parallel jobs with fairness and efficiency, J. Parallel. Distrib. Comput. 74 (3) (2014) 2180–2192, https://doi.org/10.1016/j.jpdc.2013.12.003.

[39] H. Sun, et al., Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling, in: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2018, https://doi.org/10.1109/IPDPS.2018.00029.

[40] D.E. Golberg, Genetic Algorithms In Search, Optimization, And Machine Learning, 1989, Addison Wesley, 1989, p. 36.

[41] J. Gholami, et al., Facility maintenance scheduling for organisations with a multi-location structure: optimisation model and hybrid metaheuristic approach, Int. J. Indust. Syst. Eng. 44 (1) (2023) 96–117, https://doi.org/10.1504/IJISE.2023.130962.

[42] E. Shadkam, S. Safari, S.S. Abdollahzadeh, Finally, which meta-heuristic algorithm is the best one? Int. J. Decis. Sci., Risk Manage. 10 (1-2) (2021) 32–50, https://doi.org/10.1504/IJDSRM.2021.117555.

[43] K. Asghari, et al., Multi-swarm and chaotic whale-particle swarm optimization algorithm with a selection method based on roulette wheel, Expert. Syst. 38 (8) (2021) e12779, https://doi.org/10.1111/exsy.12779.

[44] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32, https://doi.org/10.1023/A:1010933404324.

[45] S. Del Río, et al., On the use of mapreduce for imbalanced big data using random forest, Inf. Sci. 285 (2014) 112–137, https://doi.org/10.1016/j.ins.2014.03.043.

[46] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (Mar) (2003) 1157–1182.

[47] T.T. Nguyen, et al., Heterogeneous classifier ensemble with fuzzy rule-based meta learner, Inf. Sci. 422 (2018) 144–160, https://doi.org/10.1016/j.ins.2017.09.009.

[48] Workflows, P. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator. 2023.

[49] H. Gholami, M.T. Rezvan, A memetic algorithm for multistage hybrid flow shop scheduling problem with multiprocessor tasks to minimize makespan, Int. J. Indust. Eng. Manage. Sci. 7 (1) (2020) 127–145, https://doi.org/10.22034/IJIEMS.2020.118105.

[50] S. Zhang, T. Wong, Integrated process planning and scheduling: an enhanced ant colony optimization heuristic with parameter tuning, J. Intell. Manuf. 29 (2018) 585–601, https://doi.org/10.1007/s10845-014-1023-3.

Hadi Gholami obtained his BSc and MSc degrees in Computer Engineering from Islamic Azad University, Iran. His main research interests are in algorithms, scheduling, parallel and distributed computing, and machine learning.

Hongyang Sun is an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Kansas, USA. Previously, he held a research faculty position at Vanderbilt University, USA, and was a postdoctoral researcher at ENS de Lyon, INRIA (Rhône-Alpes), and IRIT (Toulouse), France. He obtained his Ph.D. in Computer Science from Nanyang Technological University, Singapore. His research interests are in the broad area of high-performance computing (HPC), cloud/edge computing, computational data science, scheduling algorithms, and fault tolerance.