Genetic Programming for Dynamic Workflow Scheduling in Fog Computing

Meng Xu, *Student Member, IEEE*, Yi Mei, *Senior Member, IEEE*, Shiqiang Zhu, Beibei Zhang, Tian Xiang, Fangfang Zhang, *Member, IEEE*, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Dynamic Workflow Scheduling in Fog Computing (DWSFC) is an important optimisation problem with many real-world applications. The current workflow scheduling problems only consider cloud servers but ignore the roles of mobile devices and edge servers. Some applications need to consider the mobile devices, edge, and cloud servers simultaneously, making them work together to generate an effective schedule. In this article, a new problem model for DWSFC is considered and a new simulator is designed for the new DWSFC problem model. The designed simulator takes the mobile devices, edge, and cloud servers as a whole system, where they all can execute tasks. In the designed simulator, two kinds of decision points are considered, which are the routing decision points and the sequencing decision points. To solve this problem, a new Multi-Tree Genetic Programming (MTGP) method is developed to automatically evolve scheduling heuristics that can make effective real-time decisions on these decision points. The proposed MTGP method with a multi-tree representation can handle the routing decision points and sequencing decision points simultaneously. The experimental results show that the proposed MTGP can achieve significantly better test performance (reduce the makespan by up to 50%) on all the tested scenarios than existing state-of-the-art methods.**

***Index Terms*—Dynamic workflow scheduling, genetic programming, fog computing.**

I. Introduction

THE widespread use of mobile devices such as smartphones and intelligent robots brings a large number of requests and data from users. These requests and data can be abstracted as workloads, such as web applications, bags of tasks, and scientific workflows [1]. Scientific workflows contain many dependent tasks and can be used to support a variety of practical studies, such as examining the structure of galaxies [2] and searching for gravitational waves [3]. Typically, these scientific workflows cannot be executed by the limited computational and storage capabilities of the mobile device. Cloud computing [4] technology is therefore widely used for the execution of workflows. Tasks in a scientific workflow can be uploaded to be executed on cloud servers. However, the upload process involves transferring raw data to the cloud server, which produces long transmission delays and privacy issues [5]. To reduce the transmission delay, fog computing is proposed [6], [7], [8], which contains two computing layers (i.e., edge and cloud). As compared to cloud computing, fog computing brings computing resources closer to users, enhancing location-based services [9]. The purpose of fog computing is to handle part of the tasks on edge servers, rather than imposing all the tasks on the cloud [10]. However, how to allocate tasks to edge or cloud servers to make the execution process more efficient is a challenging workflow scheduling problem.

Workflow scheduling [11] is a process of mapping and organising interdependent tasks on distributed processing elements to meet important objectives such as minimising makespan, load balancing, and budget. Previous studies mainly focus on static workflow scheduling problems in which all the information regarding the servers and workflows is known in advance [12]. However, in the real world, the environment is dynamic and the stream of workflows is unpredictable [13]. The information of workflows is unknown until the workflows arrive. In addition, current research does not consider mobile devices that release workflows as computing resources. Many real-world applications require mobile devices to have the ability to make autonomous decisions. Therefore, a new fog computing paradigm considering the mobile device, edge, and cloud is required. In this article, we focus on **D**ynamic **W**orkflow **S**cheduling (DWS) with dynamic workflows arrivals and limited computing resources in **F**og **C**omputing (DWSFC).

The DWSFC problem is very challenging. First, the exact methods, such as branch-and-bound [14] and mathematical programming [15] cannot efficiently handle large-scale scenarios and/or dynamic events because of their high computational cost, although they can guarantee optimality for small-scale instances. Second, the solution optimisation heuristic methods, such as ant colony optimisation [16] and particle swarm optimisation [17] that can handle large-scale scenarios, still have the limitation of high computational cost to handle dynamic events. Scheduling heuristics have been shown as a promising technique for solving scheduling problems [18]. Scheduling heuristics are used to assign a priority for the available servers, then the server with the highest priority is selected to execute

Manuscript received 5 March 2022; revised 30 August 2022; accepted 20 February 2023. Date of publication 8 March 2023; date of current version 8 August 2023. This work was supported in part by the Marsden Fund of New Zealand Government under Grant MFP-VUW1913 and in part by the MBIE SSIF Fund under Grant VUW RTVU1914. Recommended for acceptance by B. DiMartino. *(Corresponding author: Tian Xiang.)*

Meng Xu, Yi Mei, Fangfang Zhang, and Mengjie Zhang are with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: meng.xu@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; fangfang.zhang@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Shiqiang Zhu, Beibei Zhang, and Tian Xiang are with the Intelligent Robotics Research Center, Zhejiang Lab, Hangzhou 311121, China (e-mail: zhusq@zhejianglab.com; bzeecs@gmail.com; txiang163@163.com).

Digital Object Identifier 10.1109/TSC.2023.3249160

the ready task (i.e., the task with all the preceding tasks completed). Scheduling heuristics can make decisions based on the latest information and can react in real-time. The state-of-the-art scheduling heuristics for task/workflow scheduling problems include HEFT, FCFS, MaxMin, and MinMin, which have been widely used in the cloud and fog computing industry. However, manually designing scheduling heuristics is time-consuming and needs domain knowledge. The scheduling heuristics designed by humans might not capture all the important factors and complex interactions between them, and the decisions made by manually designed scheduling heuristics are mostly greedy decisions, which might not be good ones in the long-term scheduling process. The existing scheduling heuristics are more likely to be ineffective when the problem (e.g., objective and constraints) changes, particularly with dynamic nature. Thus, an automatic design method is needed to learn scheduling heuristics.

Genetic Programming Hyper-Heuristic (GPHH) [19] has been widely used to automatically generate scheduling heuristics for many combinatorial optimisation problems, including job shop scheduling [20], [21], [22], [23], bin packing [24], [25], [26] and routing problems [27], [28], [29]. Some recent research has also attempted to develop GPHH for solving DWS problems in cloud computing [30], [31]. However, the existing studies still have limitations. First, mobile devices are not included in the computation network, that is, mobile devices cannot make autonomous decisions. Second, only the cloud servers are used as computing resources, while the mobile devices and edge servers are not considered. Third, they assume an unlimited number of computing resources, which are not available in real-world applications. Lastly, they only use GPHH to evolve a rule for selecting a processor for each task, while still using the manual rule to sequence the tasks.

To solve the DWSFC problem, we design a new problem model and a corresponding simulator to imitate the scheduling process. A novel GPHH method is then proposed to solve the DWSFC problem by handling the processor assignment and task sequencing simultaneously. Specifically, this article has the following contributions.

1) A new problem model taking mobile device, edge, and cloud into consideration is presented. This new problem model takes real-world constraints (limited computing resources) into consideration and gives a novel computing paradigm with the mobile device, edge, and cloud, simultaneously.
2) A new simulator is developed to imitate the scheduling process. In this new simulator, mobile devices release workflows and decide whether to upload the tasks to edge/cloud. Mobile devices, edge/cloud all have processing ability, which are seen as computing resources.
3) A realistic circumstance of limited computing resources (edge and cloud servers) is considered. In this case, two kinds of decision points are designed, one is the routing decision point (when a task is ready), the other is the sequencing decision point (when a processor is idle). Then, a new scheduling heuristic with a routing rule (for handling the routing decision points) and a sequencing rule (for handling the sequencing decision points) is designed to account for busy missions.
4) A new MTGP method is proposed with a new representation, which has two trees, one represents the routing rule, the other denotes the sequencing rule. New terminals are designed to evolve scheduling heuristics for the DWSFC problem.
5) Experiments are performed to show the effectiveness of the proposed MTGP method (reduce the makespan by up to 50%) over existing methods under different scenarios based on the new simulator. The structure of the evolved scheduling heuristic is analysed to gain insights from the scheduling process.

The rest of this article is as follows. Section II introduces the background, including the workflow scheduling problems, and GPHH. The related work is introduced in Section III. Section IV describes the definition of the new DWSFC problem. The proposed simulation model and method are described in Sections V and VI, respectively. Section VII describes the experimental design and results. Further analyses are shown in Section VIII. Finally, Section IX concludes this article.

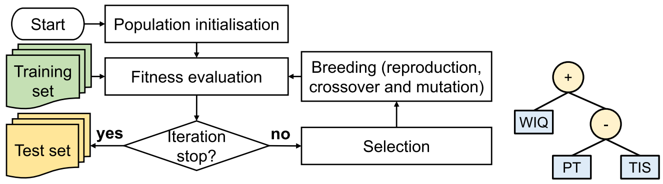
II. Background

A. Workflow Scheduling

Workflow scheduling problems can be classified into two categories according to the available information of workflows and environments, which are static problems and dynamic problems [32]. For static problems, the information about workflows and environments is known in advance. For dynamic problems, the scheduling process needs to meet some dynamic events, such as the dynamic arrival of workflows [13] and computing resources failure [33]. Dynamic arrival of workflows is the most commonly happened dynamic event in the real world. Once a task becomes ready for allocation, a scheduling heuristic will be used to select a computing resource to process it based on the real-time information [34]. There has been extensive research on static workflow scheduling problems, while the studies on dynamic workflow scheduling problems are limited. In addition, the current workflow scheduling problems are studied in cloud computing environments [32], an extension in fog computing environments is needed as the low latency requirements. Also, some studies [31] assume that the cloud provider has unlimited computing resources which are unavailable in practice. The limited computing resources will make the scheduling process busy, sometimes the uploaded tasks have to wait to be executed in the waiting queue.

Beyond that, the existing scientific workflows [1] used for workflow scheduling studies have only a few types of workflows with different Directed Acyclic Graph (DAG) structures, task data, and processing time. However, in real life, there are many different applications, which can be abstracted as many different workflows.

Therefore, in this article, we consider a more complex dynamic workflow scheduling problem with limited computing resources, in which the computing resources include the mobile device, edge, and cloud and they all have the execution ability. In



(a) Traditional GPHH (b) Tree

Fig. 1. The flowchart of the traditional GPHH method and an example of tree-based representation.

addition, we consider workflows with many different task data and processing times.

B. Genetic Programming Hyper-Heuristics

Hyper-heuristic methods aim to select or generate heuristics to efficiently tackle hard computational search problems in the heuristic space rather than the solution space [35]. GPHH, with the advantages of flexible representations, has attracted much attention [36], [37], [38]. The whole process of GPHH can be divided into two phases, which are the training process and the test process [39]. The output of the training process is a heuristic. Then, the heuristic is used as the input to the test process, which allows for measuring its performance. The flowchart of the traditional GPHH method can be seen in Fig. 1(a).

It can be seen that the training process involves four parts. In the beginning, initialisation is done to generate a population size of individuals. Each individual is randomly generated based on a designed representation. For example, the tree-based structure [40] is widely used. An example of tree-based representation is shown in Fig. 1(b). The tree is composed of terminals (e.g., WIQ, PT, TIS) and functions (e.g., $+, -$), where WIQ denotes the work of the tasks in the queue, PT represents the processing time of the task, and TIS is the time that the task has been in the system. After initialisation, the fitness evaluation process estimates the fitness of each individual by applying it to the training set. The fitness obtained by the evaluation process indicates the performance of each individual and plays a significant role in guiding the search direction. After fitness evaluation, the selection is held as the process of selecting individuals based on fitness. After a round of selection to get a well-adapted parent, the breeding process performs a genetic operator on the parents to generate offspring. The commonly used genetic operators in the evolutionary process include crossover, mutation, and reproduction.

III. Related Work

A. Simulators

Widely used simulators for task scheduling include *GridSim* [41], *CloudSim* [42], *WorkflowSim* [43] and *iFogSim* [44]. *GridSim* supports modeling and simulation of heterogeneous grid resources. Some research [45], [46] propose scheduling algorithms based on *GridSim*. Although *GridSim* is capable of modeling and simulating the Grid application behaviors in a distributed environment, it is unable to support the infrastructure and application-level requirements arising from the cloud computing paradigm [47]. *CloudSim* is implemented at the next level by programmatically extending the core functionalities exposed by the *GridSim*. *CloudSim* is proposed to model and simulate cloud computing systems and application provisioning environments. Cloud computing offers services at the infrastructure level that can scale to the Internet of Things storage and processing requirements. *WorkflowSim* extends the existing *CloudSim* simulator by providing a higher layer of workflow management. *iFogSim* is proposed to overcome the limitation of *CloudSim* with low latency. It extends *CloudSim* by including the edge of the network to decrease the latency and network congestion.

All these simulators have contributed to the research of task scheduling problems. However, as technology advances, mobile devices are expected to have high processing speeds and autonomous decision-making capabilities, which are not considered in these simulators. To this end, a novel simulator is developed in this article.

B. Methods

Currently, there are mainly three kinds of methods used for workflow scheduling problems, which are exact methods, heuristic methods, and hyper-heuristic methods.

In the early years, exact methods, such as branch-and-bound [14], [48] and mathematical programming [49] are designed for solving static task scheduling problems. These methods can obtain optimal solutions. For small-scale problems, the computational time of such methods is acceptable. However, they are not applicable for large-scale problems, due to their high computational complexities.

Heuristic methods, as an efficient way of searching for reasonably good solutions, have been used for solving task scheduling problems. Genetic algorithm [50], [51], [52] is one of the most widely studied heuristic methods for task scheduling problems. Other heuristic methods, such as particle swarm optimisation [53] and ant colony optimisation [54], have also been investigated for grid and cloud environments. These methods can obtain good performance by the evolution process. However, they still have the limitation of high computational cost and handling dynamic events [32].

Different from the above heuristic methods, scheduling heuristics have been designed to make a decision in real-time. As a greedy method, scheduling heuristics can give each candidate processor/task a priority quickly. Then, the processor/task with the highest priority is selected. Heterogeneous Earliest Finish Time (HEFT) [55], First Come First Serve (FCFS) [56], MINMIN [57] and MAXMIN [58] are manually designed scheduling heuristics which have obtained reasonable performance for task scheduling problems. However, these methods are only designed for specific scenarios and the design of effective scheduling heuristics heavily relies on domain expertise and is time-consuming. To address this issue, an automatic design method is needed to generate an effective scheduling heuristic.

TABLE I
NOTATIONS USED IN THE PROBLEM FORMULATION

Notation	Description
W_i	The i-th workflow
T_{ij}	The j-th task in the i-th workflow
$pred(T_{ij})$	The set of preceding tasks of T_{ij}
$succ(T_{ij})$	The set of succeeding tasks of T_{ij}
φ_{ij}	The workload of task T_{ij}
$D_{ij}^{\uparrow}$	All the input data of task T_{ij}
$D_{ij}^{\downarrow}$	All the output data of task T_{ij}
t_i	The release time of workflow W_i
P_k	The k-th processor
γ_k	The processing rate of P_k
$B_{k,m}$	The bandwidth between the processor P_k and the mobile device processor P_m
τ_{ijk}	The processing time of task T_{ij} on processor P_k
$\tau_{ijk}^{\uparrow}$	The upload time of all the relavent data of task T_{ij} to the processor P_k
$\tau_{ijk}^{\downarrow}$	The download time of all the relavent data of task T_{ij} from the processor P_k

In [30], GPHH was used for the first time to solve the DWS problem. In this article, each individual has one tree which is used to select a processor for each task. Each individual is evaluated based on the *WorkflowSim* simulator. In [31], new terminals and functions are designed to help GPHH get better performance than traditional GPHH. However, the current GPHH methods for DWS problems do not consider sequencing decision points which are as important as the routing decision points.

IV. PROBLEM DESCRIPTION

A. Workflow Model

The workflows $\mathcal{W}$ are released by the mobile devices over time. Each workflow $W_i \in \mathcal{W}$ is generated by a mobile device P_{m_i} at time t_i, and has a set of tasks $\mathcal{T}_i$ and their dependencies can be represented as a DAG. Specifically, each task $T_{ij} \in \mathcal{T}_i$ has a set of preceding tasks $pred(T_{ij}) \subset \mathcal{T}_i$ and a set of succeeding tasks $succ(T_{ij}) \subset \mathcal{T}_i$. A task with no preceding task is called an *entry task*, and a task with no succeeding task is called an *exit task*. A workflow is completed after all the exit tasks are completed.

Each task T_{ij} has its workload φ_{ij}. To process a task T_{ij}, certain input data $D_{ij}^{\uparrow}$ is required. The process generates output data $D_{ij}^{\downarrow}$ that serves as the input data for the succeeding tasks.

B. Processor Model

In DWSFC, there are a set of cloud servers $\mathcal{P}^c$, a set of edge servers $\mathcal{P}^e$, and a set of mobile devices $\mathcal{P}^d$. They are statically provisioned and can all be processors. Each processor $P_k \in \mathcal{P}^c \cup \mathcal{P}^e \cup \mathcal{P}^d$ has a processing rate γ_k. Each mobile device $P_m \in \mathcal{P}^d$ is linked with each edge/cloud server $P_n \in \mathcal{P}^c \cup \mathcal{P}^e$, and the bandwidth between them is denoted as $B_{m,n}$, which is assumed to be unaffected by the amount of data transfers in progress. If a task T_{ij} is allocated to be processed by an edge/cloud server P_k, then the upload time $\tau_{ijk}^{\uparrow}$, download time $\tau_{ijk}^{\downarrow}$, and processing time τ_{ijk} can be calculated as follows.

$$\tau_{ijk}^{\uparrow} = \frac{D_{ij}^{\uparrow}}{B_{m_i,k}}, \quad \tau_{ijk}^{\downarrow} = \frac{D_{ij}^{\downarrow}}{B_{m_i,k}}, \quad \tau_{ijk} = \frac{\varphi_{ij}}{\gamma_k}. \tag{1}$$

If a task T_{ij} is to be processed by the mobile device P_{m_i} itself, then $\tau_{ijm_i}^{\uparrow} = \tau_{ijm_i}^{\downarrow} = 0$.

C. Constraints and Assumptions

The goal of DWSFC is to find a schedule with the allocation of each task of each workflow to the processors, and the start time of each task on its processor, subject to the following constraints and assumptions.

- The information of each workflow is not known until it is released.
- A task can be processed only after all the preceding tasks have been completed and all the input data for the task has been transferred to the processor.
- Scheduling process is non-preemptive, that is, the processing of a task cannot be stopped or suspended once it is started.
- Each task can only be processed by the edge/cloud servers or the mobile device that releases it, while can not be processed by other mobile devices due to privacy and security concerns.
- Each processor can process up to one task at a time.
- The inherent characteristics of each processor, that is the processing rate and bandwidth, are not changed throughout the whole work, and the processor will not break down.

Note that, in practice, there is a chance of other dynamic events happening, such as disconnection, which could affect the execution time of a workflow. However, it would not happen frequently. Since this study mainly focuses on the event of dynamic workflow arrivals, we ignore this situation to simplify the problem.

D. Objective

The objective is to minimise the makespan, which is calculated as follows:

$$\texttt{makespan} = \max\{\Gamma_i \mid W_i \in \mathcal{W}\} - r_0, \tag{2}$$

where Γ_i is the completion time of the workflow W_i in the schedule and r_0 denotes the release time of the first released workflow W_0. The notations used in the problem formulation are shown in Table I.

V. THE NEW SIMULATOR

A. The Main Framework

The main framework of the proposed simulator can be seen in Fig. 2. This simulator has three main parts which are the device center, edge center, and cloud center.

In the device center, each mobile device can be seen as a relatively independent system that shares common edge/cloud resources but makes its own decisions. Each mobile device can release new workflows, store existing workflows, schedule tasks,

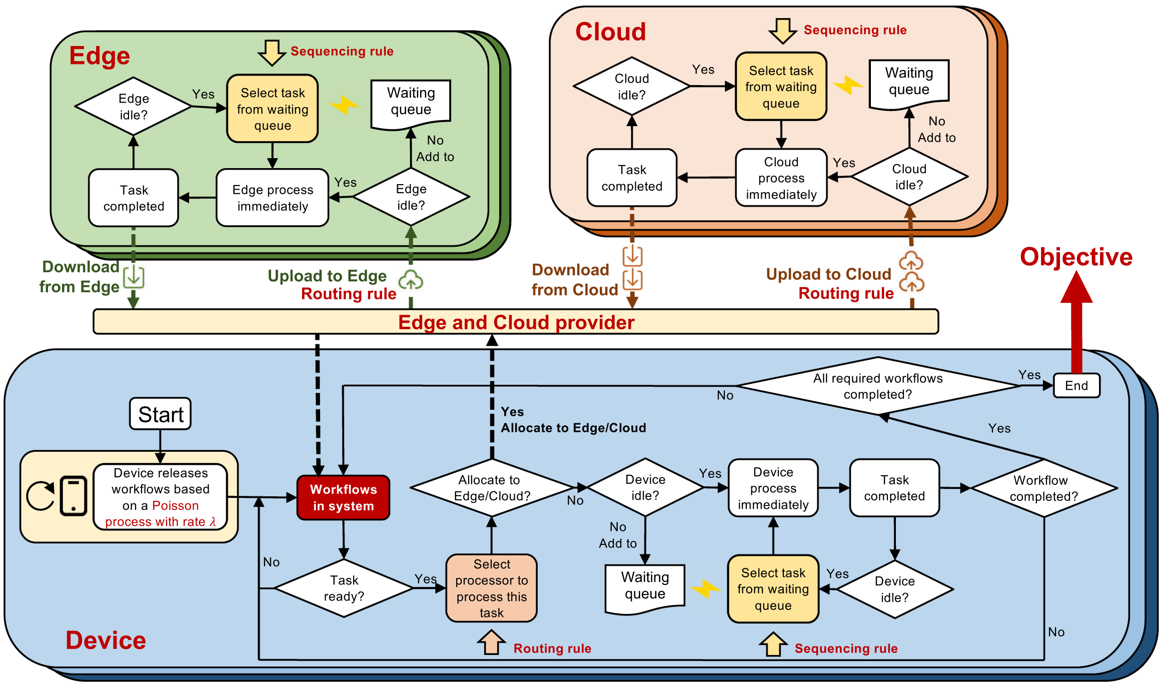


Fig. 2. The framework of the proposed simulator.

process tasks, upload data, and download data. Note that each mobile device can only process tasks that are released by itself but not those of other mobile devices. At the beginning of the scheduling process, the mobile device releases the workflow with the DAG structure. The mobile device keeps checking for ready tasks. Once a task becomes ready (routing decision point), the routing rule will be used by the mobile device to select the processor (a cloud/edge server, or the device itself) to process the task. If the decision is to process the task by the mobile device itself, the mobile device will check if it is idle. If it is idle, the mobile device will process the task immediately. If not, the task will be put into the waiting queue and waiting to be processed later. Once the mobile device becomes idle (sequencing decision point), it will use the sequencing rule to select a task from the waiting queue to process as the next task.

If the decision is to upload the task to one of the edge or cloud servers, then a similar process will be done after the task is uploaded to the selected edge/cloud server. After all the workflows are completed, the simulation process is stopped and the objective value is calculated from all the completed workflows. The detailed execution process is described in the next subsection.[1]

B. The Execution Process of Scheduling Heuristic on the Simulator

The simulation is designed as a discrete event-driven simulation process. It consists of workflow arrival event (`WorkflowArrivalEvent`), task visit event (`TaskVisitEvent`), process start event (`ProcessStartEvent`), and process finish event (`ProcessFinishEvent`). Each event has its trigger time, and an event can generate and/or trigger other events. The scheduling heuristic makes a decision on each decision point to make the scheduling process continue. In this article, a scheduling heuristic has a routing rule and a sequencing rule. The routing rule has the ability to handle the routing decision points, while the sequencing rule is used for the sequencing decisions.

The simulation maintains an event queue, which is represented as a priority queue of events (where the trigger time is the priority) as shown in (3).

$$\Delta = [\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}_e, \cdots]. \tag{3}$$

Each event is represented as $\mathcal{E}_e = (\alpha_e, W_e, T_e, P_e, \tau_e)$, where $\alpha_e \in \{0, 1, 2, 3\}$ is the event type, and W_e, T_e and P_e are the workflow, task and processor involved in this event. τ_e is the trigger time of the event.

- `WorkflowArrivalEvent` ($\alpha_e = 0$): the workflow W_e is released by a mobile device at time τ_e. It can be denoted as $(0, W_e, -, -, \tau_e)$, where "$-$" means empty value (null).
- `TaskVisitEvent` ($\alpha_e = 1$): the task T_e of the workflow W_e is arrived at its selected processor P_e at time τ_e. It can be denoted as $(1, W_e, T_e, P_e, \tau_e)$.
- `ProcessStartEvent` ($\alpha_e = 2$): the processor P_e starts to process the task T_e of the workflow W_e at time τ_e. It can be denoted as $(2, W_e, T_e, P_e, \tau_e)$.
- `ProcessFinishEvent` ($\alpha_e = 3$): the processor P_e finishes processing the task T_e of the workflow W_e at time τ_e. It can be denoted as $(3, W_e, T_e, P_e, \tau_e)$.

Normally, each workflow will go through the process as shown in Fig. 3 once it is released.

[1]The source code of the simulation, which is implemented in JAVA, can be found in https://github.com/MengBIT/Fog-Computing/tree/multiDeviceDebug.

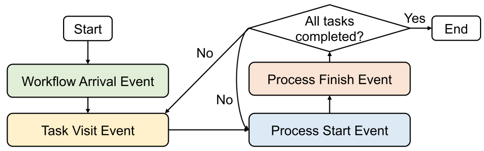


Fig. 3. The flowchart of the workflow execution process.

Algorithm 1: DWSFC Simulation.

```
   Input: A DWSFC instance, a routing rule h_r(·), a sequencing
          rule h_s(·)
   Output: A DWSFC schedule
1  Set the schedule ρ = {}, event queue Ω = {};
2  foreach W_i ∈ 𝒲 do
3  |   Create a WorkflowArrivalEvent ℰ_i and Ω = Ω ∪ ℰ_i;
4  end
5  while Ω is not empty do
6      Get the next event ℰ_e from Ω;
7      if α_e = 0 then
           // Trigger the WorkflowArrivalEvent
8          Calculate the priority h_r(P_k, T) of each processor P_k
             for each ready task T by the routing rule h_r(·);
9          Select the processor P* with the highest priority and
             create a TaskVisitEvent ℰ and Ω = Ω ∪ ℰ;
10     else if α_e = 1 then
           // Trigger the TaskVisitEvent
11         if P* is not idle then
12         |   Add T to the waiting queue of P*;
13         else
14         |   Create a ProcessStartEvent ℰ and Ω = Ω ∪ ℰ;
15         end
16     else if α_e = 2 then
           // Trigger the ProcessStartEvent
17         Calculate the processing τ and download time τ↓;
18         Create a ProcessFinishEvent ℰ and Ω = Ω ∪ ℰ;
19     else if α_e = 3 then
           // Trigger the ProcessFinishEvent
20         if the queue of P* is not empty then
21             Calculate the priority h_s(T, P*) of each task T in
                 queue by the sequencing rule h_s(·);
22             Select the task T* with the highest priority and
                 create a ProcessStartEvent ℰ and Ω = Ω ∪ ℰ;
23         end
24         if W* is not completed then
25             Calculate the priority h_r(P_k, T) of each processor
                 P_k for each ready task T of workflow W* by the
                 routing rule h_r(·) ;
26             Select the processor P* with the highest priority
                 and create a TaskVisitEvent ℰ and Ω = Ω ∪ ℰ;
27         end
28     end
29 end
30 return the obtained schedule ρ;
```

The execution process of the scheduling heuristic on the proposed simulator is shown in Algorithm 1. The DWSFC simulation starts once the first workflow arrived (line 3). The simulation continues until all the workflows are completed or the event queue Δ is empty (line 5). The scheduling process goes on by triggering each event based on its event type (line 6). When triggering the `WorkflowArrivalEvent`, the tasks without preceding tasks are ready tasks, the routing rule will be used to calculate the priority of each processor for each ready task (line 8). When triggering the `ProcessFinishEvent`, after the current task is completed by the processor, the processor becomes idle. If the waiting queue of the idle processor is not empty, the sequencing rule will be used to calculate the priority of each task for the processor to process next (line 21). Otherwise, the processor stays idle. Additionally, the completed task can make some successor tasks ready to be processed. In this case, the routing rule is used to calculate the priority of each processor for each ready task (line 25).

Based on the above description, the differences between the *iFogSim* simulator and the developed simulator are as follows. First, in the *iFogSim*, the edge layer is an intermediate network connecting the mobile devices and the cloud layer. However, in the new simulator, both the edge layer and the cloud layer communicate directly with the mobile devices, which improves the fault-tolerance of the network and enables autonomous decision-making by mobile devices. Second, in the traditional *iFogSim*, the mobile devices release workflows and transmit them to the edge/cloud layer for processing, the mobile devices do not have the processing ability. In the new simulator, mobile devices can also process the jobs themselves.

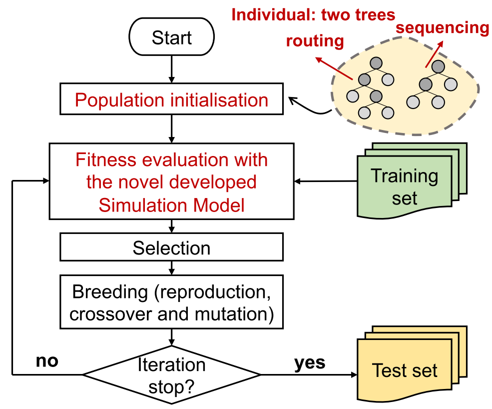


Fig. 4. The flowchart of the MTGP method.

VI. The New GP Approach

A. The Overview

The overview of the Multi-Tree Genetic Programming (MTGP) method is shown in Fig. 4. Different from the traditional GPHH methods for workflow scheduling problems, the main innovation in this article is the new representation of the individual, the fitness evaluation based on the new proposed simulator, and the new terminals related to the DWSFC problem. In this article, the routing rule and the sequencing rule share the same terminal set. The proposed MTGP is expected to automatically extract important terminals from the terminal set for the routing rule and the sequencing rule, respectively.

Apart from these, the MTGP method has the same process as the traditional GPHH, including population initialisation, fitness evaluation, selection, and breeding. The pseudo-code of MTGP can be seen as Algorithm 2 and the details about each process are shown as follows.

1) Representation: To evolve a scheduling heuristic with two rules for the DWSFC problem, an individual is designed with two

Algorithm 2: Pseudo-Code of MTGP.

```
   // Population initialisation
1  while N_ind < Popsize do
2      foreach Individual do
3          Initialise the tree for routing rule and sequencing rule
             by ramp half-and-half;
4      end
5  end
6  while Stopping criteria not met do
       // Fitness evaluation
7      Evaluate the individuals based on the proposed simulator
         as Algorithm 1;
       // Elitism selection
8      Copy the elites to the new population;
       // Parent selection
9      Select individuals based on fitness value;
       // Breeding
10     Generate offspring by applying
         crossover/mutation/reproduction operators;
11 end
12 return the best individual (scheduling heuristic);
```

trees, one for the routing rule and the other for the sequencing rule. The structure of the individual can be seen in Fig. 4.

2) Initialisation: In the beginning, a number of individuals are initialised by random selecting and combining the terminals and functions with the *ramped-half-and-half* method (line 4) [59]. That is, half of the individuals are initialised with the maximum depth set in advance, while the other half of the individuals are initialised randomly within the maximum depth.

3) Fitness Evaluation: Given the training instance, we can evaluate the fitness of an individual by applying it to the execution process as shown in Algorithm 1 (line 7). After fitness evaluation, each individual has a fitness that represents its quality.

4) Selection: The proposed MTGP method uses the classical tournament selection [60] to select parents for genetic operators (line 9). First, a set of individuals are sampled from the population randomly as candidates. Then, the individual with the best fitness is selected as the parent.

5) Breeding: The proposed MTGP uses reproduction, subtree mutation, and tree swapping crossover operators [20] (line 10). For reproduction, the selected parent based on tournament selection is directly inherited to the next generation. For subtree mutation, a new subtree is randomly generated by selecting and combining terminals and functions. Then we randomly select a node from the parent and replace the subtree under the node with the newly generated subtree. For tree swapping crossover, for one tree, we randomly select nodes from each parent and then swap the subtrees under the nodes. For the other tree, we simply swap the whole tree. The process of subtree mutation and tree swapping crossover can be seen in Figs. 5 and 6.

B. The Designed Terminals

In this article, ten terminals, which are set as the features that indicate the characteristics related to the processors, tasks, and workflows. The terminal set of MTGP is shown in Table II and the detailed description is as follows.

NIQ represents how many tasks are waiting to be processed in the waiting queue of the processor. WIQ denotes the total processing time that all the tasks in the waiting queue would cost by this processor. MRT is the earliest idle time after finishing the current process. TTIQ represents the total processing time plus the upload and download time that all the tasks in the waiting queue would cost by this processor. UT denotes the upload time of the task. DT represents the download time of the task. PT represents the processing time of the task. TWT is the waiting time of the task. TIS represents how much time the workflow has been stored in the system when it is released. NTR denotes how many tasks have not been processed in the workflow.

The above terminals are extracted from the scheduling process which can describe the system state clearly.

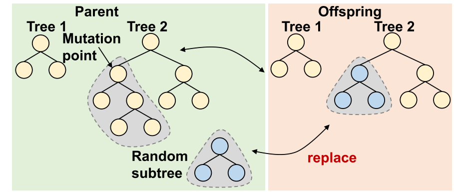


Fig. 5. The process of the subtree mutation.

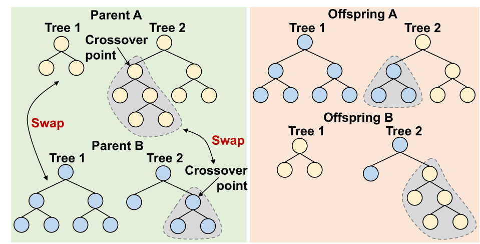


Fig. 6. The process of the tree swapping crossover.

TABLE II
THE TERMINAL SET

Notation	Description
NIQ	The number of tasks in the waiting queue.
WIQ	The remaining work in the waiting queue.
MRT	The ready time of server/device.
UT	The upload time of the task.
DT	The download time of the task.
PT	The processing time of the task.
TTIQ	The total remaining time in the waiting queue.
TIS	The time in system: t - releaseTime.
TWT	The waiting time of the task.
NTR	The number of tasks remaining of the workflow.

TABLE III
DIFFERENT TYPES OF WORKFLOW

Type	Number of tasks		
	Class A	Class B	Class C
CyberShake	30	50	100
Epigenomics	24	46	100
Inspiral	30	50	100
Montage	25	50	100
Sipht	30	60	100

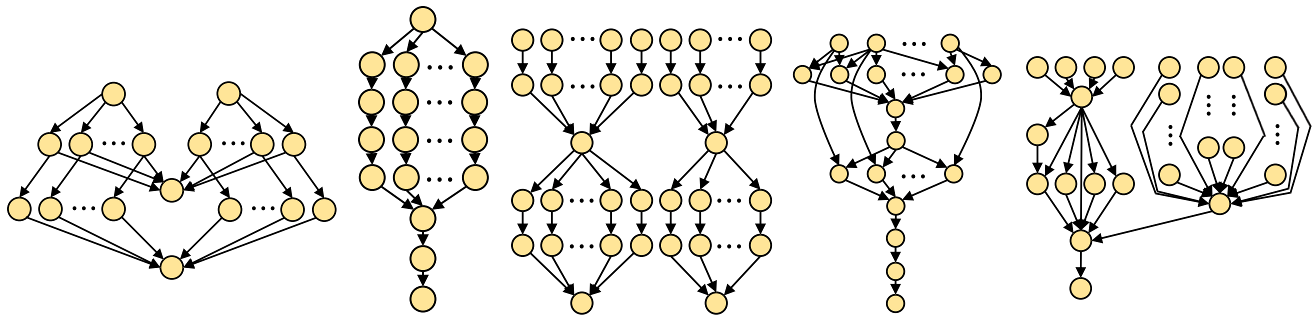


Fig. 7. The structure of five types of workflow.

TABLE IV
DIFFERENT TYPE OF SERVERS

type	bandwidth	processing rate
mobile device	-	[125, 250]
fog server	1024	[250, 500]
cloud server	128(20%)/256(80%)/512(20%)	[500,1000]

TABLE V
THE SCENARIOS OF DIFFERENT SCALES

Scenario	Num*	Workflow type	Mobile device	Edge/ Cloud
small 1	10/20	Class A	1	20/20
small 2	10/20	Class A	2	20/20
small 3	10/20	Class A	3	20/20
medium 1	15/30	Classes A and B	1	30/30
medium 2	15/30	Classes A and B	2	30/30
medium 3	15/30	Classes A and B	3	30/30
large 1	25/50	Classes A, B, and C	1	60/60
large 2	25/50	Classes A, B, and C	2	60/60
large 3	25/50	Classes A, B, and C	3	60/60

* The number of warm-up workflows/considered workflows.

VII. EXPERIMENT STUDIES

A. Datasets

For the new simulator, workflows will be released by the mobile device over time according to a Poisson process [11], [13]. Each workflow has a different structure and a different number of tasks as shown in Table III. The workload of each task is randomly generated with the range [5000,15000]. The input and output data of each task is assigned by a uniform discrete distribution between 5,220 and 20,680. The information about processors, including processing rate and bandwidth, are randomly initialised according to Table IV.

We design nine scenarios with three kinds of scales, which cover small, medium, and large, as shown in Table V. The structure of all the workflow types can be seen in Fig. 7 and Table III gives the number of tasks of workflow for different classes (A, B, and C). The small-scale scenarios consider 20 workflows with the number of tasks from class A. The medium scenarios consider 30 workflows with the number of tasks from classes A and B. The large scenarios consider 50 workflows with the number of tasks from classes A, B, and C. At the same time, for the same scale scenario, we consider different numbers of the mobile device from 1 to 3. In addition, for each scenario, the half number of workflows considered in the scenario is used as warm-up workflows to obtain a stable scheduling system. The simulation stops when the warm-up workflows and the considered workflows are completed.

TABLE VI
THE PARAMETER SETTINGS OF MTGP

Parameter	Value
Population size	1000
Number of generations	51
Method for initialising population	ramped-half-and-half
Initial minimum/maximum depth	2 / 6
Elitism	10
Maximal depth	8
Crossover rate	0.80
Mutation rate	0.15
Reproduction rate	0.05
Parent selection	Tournament selection
Terminal/non-terminal selection rate	10% / 90%

B. Parameter Setting

The set of functions is as $\{+, -, \times, \div, max, min\}$. The arithmetic operators take two arguments. The "$\div$" operator is protected and returns 1 if divided by zero. The "max" and "min" functions take two arguments and return the maximum and minimum of their arguments, respectively. The other parameters of MTGP are shown in Table VI.

C. Test Performance

The effectiveness of the proposed MTGP algorithm is verified with a comparison to seven manually designed scheduling heuristics [55], [61], [62], [63], [64]. In order to make the seven scheduling heuristics more suitable for solving DWSFC problems with the objective of minimising the makespan, their scheduling principles are listed as follows:

- HEFT [55]: at each routing decision point, it selects the processor with the earliest execution finish time. At each sequencing decision point, it selects the task with the highest upward rank which is the length of the critical path from the task to an exit task.

TABLE VII
THE MEAN (STANDARD DEVIATION) RESULTS OF TEST PERFORMANCE OF 30 INDEPENDENT RUNS OF MTGP AND BASELINE METHODS FOR NINE SCENARIOS

Scenario	Algorithm							
	HEFT	FCFS	MaxMin	MinMin	SDLS	BWAWA	CEAS	MTGP
small 1	4061.98(0)	4073.99(0)	5069.91(0)	4718.63(0)	4195.05(0)	26702.13(0)	23665.55(0)	2397.09(65.63)(-)
small 2	3943.61(0)	3208.41(0)	5008.89(0)	4651.75(0)	4019.07(0)	21381.84(0)	15179.14(0)	1702.01(80.56)(-)
small 3	4625.65(0)	2802.89(0)	5160.01(0)	4724.87(0)	4508.39(0)	16348.69(0)	10932.41(0)	1491.17(83.16)(-)
medium 1	6037.21(0)	5789.95(0)	7177.32(0)	6778.37(0)	6015.81(0)	55420.12(0)	50289.77(0)	3636.68(463.95)(-)
medium 2	5546.62(0)	4574.06(0)	7223.89(0)	7008.76(0)	5802.42(0)	42186.26(0)	31948.47(0)	2265.32(66.98)(-)
medium 3	6967.15(0)	4376.07(0)	8183.54(0)	7928.81(0)	7260.67(0)	31847.55(0)	22540.78(0)	1930.73(68.85)(-)
large 1	9815.00(0)	9323.27(0)	11581.32(0)	11500.43(0)	9485.03(0)	136872.23(0)	126783.45(0)	6242.00(876.82)(-)
large 2	9404.16(0)	7344.98(0)	12350.29(0)	12219.01(0)	9488.37(0)	89333.21(0)	78027.79(0)	3562.01(189.46)(-)
large 3	10066.59(0)	6514.15(0)	13701.21(0)	12996.36(0)	10235.10(0)	66391.26(0)	56011.41(0)	2633.26(113.09)(-)

- FCFS [61]: at each routing decision point, it selects the first idle processor. At each sequencing decision point, it selects the first arrived task.
- MAXMIN [61]: at each routing decision point, it selects the first idle processor. At each sequencing decision point, it selects the task with the longest processing time.
- MINMIN [61]: at each routing decision point, it selects the first idle processor. At each sequencing decision point, it selects the task with the shortest processing time.
- SDLS [62]: at each routing decision point, it selects the processor with the highest stochastic dynamic level, defined as the task's stochastic upward rank minus the task's earliest execution start time, plus the varying computation capacity of the processor. At each sequencing decision point, it selects the task with the highest stochastic upward rank.
- BWAWA [63]: it is designed to minimise the makespan without affecting the increase in energy consumption. To make it focus on minimising makespan, this article modifies it by ignoring the features related to energy in the priority function. At each routing decision point, it selects the processor with the shortest transport time. At each sequencing decision point, it selects the task with the lowest downward rank which is the length of the critical path from an entry task to the task.
- CEAS [64]: it is proposed to minimise the execution cost and reduce energy consumption. To make it focus on minimising makespan, we modify it by ignoring the features related to execution cost and energy in the priority function. At each routing decision point, it selects the processor with the shortest processing time plus the longest transport time. At each sequencing decision point, it selects the task with the earliest release time plus the shortest processing time.

For the MTGP algorithm, 30 independent runs are done for each scenario and the evolved scheduling heuristics are tested on 30 instances. A Wilcoxon rank-sum test with a significance level of 0.05 is then used to validate the performance of the proposed algorithm [65]. The "−/+/=" indicates that the corresponding result is significantly better than, worse than, or similar to the comparison algorithm.

As listed in Table VII, we can conclude that the proposed MTGP method performs significantly better than all the baseline methods on all nine scenarios. The makespan obtained by the baseline methods is basically two or three times worse than the proposed MTGP method.

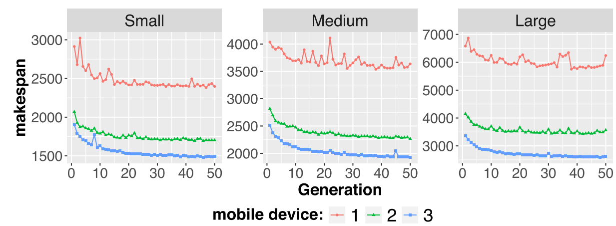


Fig. 8. Convergence curves of test fitness on nine scenarios.

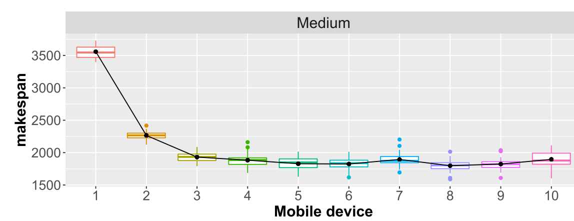


Fig. 9. Box plots and curve of test fitness on ten medium scenarios.

Fig. 8 gives the convergence curves of the proposed MTGP method. Based on these convergence curves, we can see some phenomena. First, as the scale of the scenario increases, the makespan is gradually increasing. This is because large-scale scenarios consider more workflows and workflows often contain more tasks, which leads to high execution time. Second, in the early stage, the convergence speed is faster and after about 20 generations, the convergence speed decreases and tends to level off after about 40 generations. Third, as the evolutionary process goes on, the results converge incrementally, and even if there are fluctuations in the middle of the process, eventually it converges to a good and stable solution.

D. Analysis About the Number of Mobile Devices

In this section, we analyse the influence of the number of mobile devices by testing the proposed method on ten medium scenarios with different numbers of mobile devices. As seen from Fig. 9, when other hyperparameters are fixed and only the number of mobile devices varies (increase from 1 to 10), the makespan decreases. In the early stages, the decline in makespan is dramatic (in particular, the number of mobile devices changes from 1 to 2), while in the later stages the decline tapers off and flattens out. Although more mobile devices mean more intensive

TABLE VIII
THE EVOLVED ROUTING RULE AND SEQUENCING RULE SIZES

Scenarios	routing rule size		sequencing rule size	
	min	mean(std)	min	mean(std)
small 1	31.00	57.60(18.93)	1.00	16.53(9.03)
small 2	27.00	52.47(16.97)(=)	1.00	17.53(13.15)(=)
small 3	31.00	60.27(13.72)(=)(+)	1.00	17.27(11.81)(=)(=)
medium 1	21.00	55.53(15.57)	3.00	13.93(7.44)
medium 2	29.00	55.67(16.33)(=)	3.00	18.33(13.91)(=)
medium 3	25.00	57.47(16.43)(=)(=)	1.00	22.60(15.30)(+)(=)
large 1	25.00	47.27(18.27)	3.00	18.20(11.30)
large 2	25.00	50.60(14.65)(=)	1.00	21.33(15.86)(=)
large 3	29.00	55.40(20.70)(=)(=)	5.00	22.20(15.12)(=)(=)

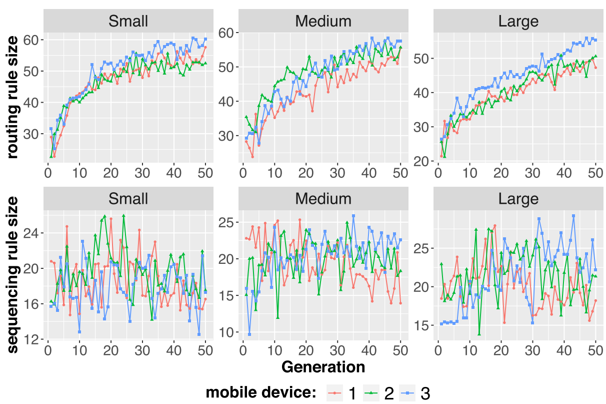


Fig. 10. Curves of routing and sequencing rule size on nine scenarios.

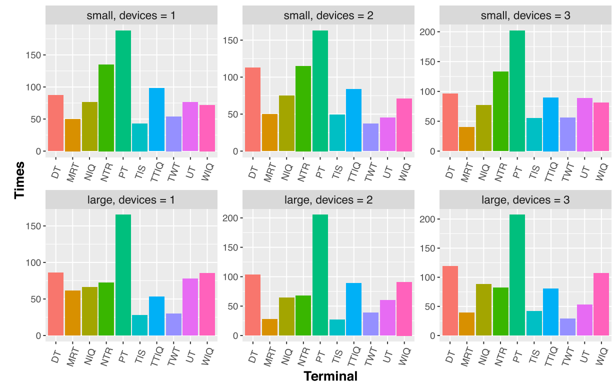


Fig. 11. Frequency of terminals in routing rules.

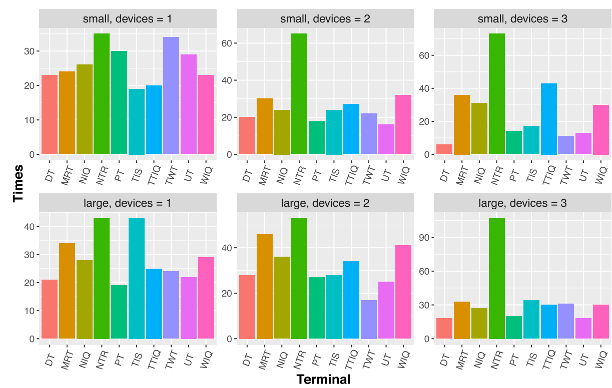


Fig. 12. Frequency of terminals in sequencing rules.

workflow arrival times, the processors without upload/download time (the mobile devices) are added to the system which can help improve the overall processing speed. The MTGP method can learn from the evolutionary process to evolve a good scheduling heuristic for this situation.

However, for the baseline methods, like the HEFT, MAXMIN, and MINMIN, the scheduling principle is not changed when the scenarios vary. They use the same policy for all the scenarios, so increasing the number of mobile devices can even increase the makespan, as shown in Table VII. Therefore, we can see that the manually designed scheduling heuristics cannot handle different scenarios well. On the other hand, the proposed MTGP method can obtain good performance on different scenarios with the evolved scheduling heuristics.

VIII. FURTHER ANALYSES

A. Rule Size

A smaller rule (fewer nodes in the tree) tends to have better interpretability [66]. The mean (standard deviation) of routing rule size and sequencing rule size of 30 independent runs of MTGP for nine scenarios are shown in Table VIII. For the three scale scenarios, the latter results are compared with the former results based on a Wilcoxon rank-sum test with a significance level of 0.05. Fig. 10 shows the convergence curves of routing rule size and sequencing rule size.

Based on Table VIII and Fig. 10, we can see that, for all the scenarios, the routing rules have larger sizes than the sequencing rules (more than 2 times). More specific analysis results show that, for the small-scale scenarios, the routing rules on the small 1 scenario have significantly smaller rule sizes than that on the small 3 scenario, For the medium-scale scenarios, the sequencing rules on the medium 1 scenario have significantly smaller rule sizes than that on the medium 3 scenarios. For the large-scale scenarios, both the size of the routing rules and sequencing rules are not sensitive to the number of mobile devices.

In summary, we can see that the routing rules evolved for DWSFC are usually more complex than the sequencing rules. Neither the routing rule sizes nor the sequencing rule sizes are sensitive to the number of mobile devices on most of the scenarios.

B. Feature Analysis

Figs. 11 and 12 show the number of times each terminal is used in the routing rules and sequencing rules from the same evolved scheduling heuristics in the 30 runs on small-scale and large-scale scenarios. Based on these results, the following observations can be obtained.

- Most terminals are used more frequently in the routing rules than that in the sequencing rules.
- For the routing rule, PT is the most frequently used terminal among the 10 terminals, indicating that the processing time of processors plays a key role when selecting the processor

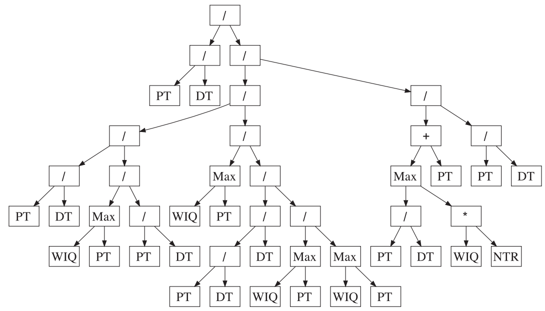


Fig. 13. An example of routing rule.

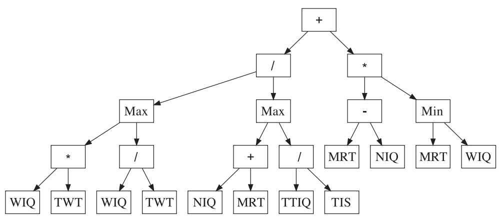


Fig. 14. An example of sequencing rule.

for the ready tasks. In addition, DT, NTR, TTIQ, UT, and WIQ are also important criteria for the selection of a processor.

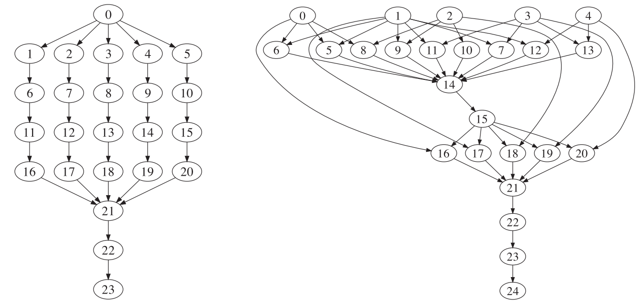
- For the sequencing rule, when there is only one mobile device, most of the terminals are used frequently. When there are two or three mobile devices, the distribution of terminal frequency is different from the scenarios with one mobile device, NTR is the most frequently used terminal among the 10 terminals, and DT is used less frequently, especially on the small 3 scenario.
- The distribution of the terminal frequency is different between small and large scenarios, even with the same number of mobile devices. For example, for the routing rule, the terminal usage frequency of NTR, TIS, and TWT in large-scale scenarios is smaller than that in small-scale scenarios.

In summary, it can be seen that PT and DT play very important roles on the routing rule, while these two terminals are not used frequently on the sequencing rule. However, NTR is important criteria for the sequencing rule.

C. Structure Analysis of the Evolved Scheduling Heuristic

To further understand the behaviour of the scheduling heuristic evolved by the proposed MTGP, an evolved scheduling heuristic is selected to do further analysis. Figs. 13 and 14 show a routing rule and a sequencing rule which come from the same scheduling heuristic. The selected scheduling heuristic has promising test performance.

In terms of the routing rule, it is a combination of four terminals (DT, PT, WIQ, and NTR), where PT is the most frequently used terminal in this rule, which has been used 11 times. It is followed by DT which is used 7 times and WIQ which is used 5 times. NTR, on the other hand, is used only once. Additionally, the subtree $\frac{PT}{DT}$ is used 6 times, and the subtree $\max\{WIQ, PT\}$ is used 4 times. The routing rule can be simplified to R_0 as shown in (4).



(a) The workflow 0. (b) The workflow 1.

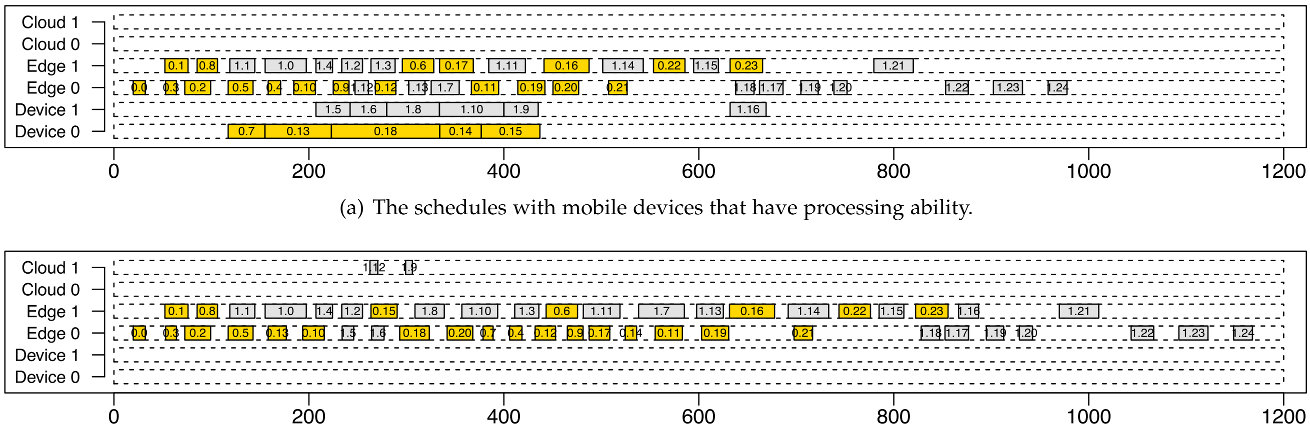
Fig. 15. The DAG structures of two workflows.

$$R_0 = \frac{DT^4}{PT^3} \times \max\{WIQ, PT\}^2 \times \max\left\{\frac{PT}{DT}, WIQ \times NTR\right\}. \quad (4)$$

This rule suggests that if there are many tasks remaining in the waiting queue, i.e., the work remaining (WIQ) is larger than the processing time (PT) of the current ready task, then this routing rule can be further simplified as $R_1 \approx \frac{DT^4 \times WIQ^2 \times NTR}{PT^3}$, where NTR (the number of tasks remaining of the workflow this ready task belongs to) is determined by the ready task. Because NTR will not be changed by the candidate processors, the routing rule means that the selection strategy is mainly based on the download time (DT), work remaining in the waiting queue (WIQ), and the processing time (PT) of the processors. If the work remaining (WIQ) of the server is smaller than the processing time (PT) of the current ready task or there is no task remaining in the waiting queue, this routing rule can be further simplified as $R_2 \approx DT^3$, which means that the selection strategy is mainly based on the download time (DT) of the processors. In other words, this routing rule indicates that the processor with a smaller download time and smaller work remaining is preferred. This is almost consistent with our intuition that the processors which are not busy and have short transportation time are good choices.

In terms of the sequencing rule, it is a combination of six terminals (WIQ, MRT, NIQ, TWT, TTIQ, and TIS), where WIQ and MRT are the most frequently used terminals in this rule, which are both used 3 times. The sequencing rule can be simplified to S_0 as shown in (5).

$$\begin{aligned} S_0 &= \frac{\max\{WIQ \times TWT, \frac{WIQ}{TWT}\}}{\max\{NIQ + MRT, \frac{TTIQ}{TIS}\}} + (MRT - NIQ) \\ &\quad \times \min\{MRT, WIQ\} \\ &= \frac{WIQ \times TWT}{\max\{NIQ + MRT, \frac{TTIQ}{TIS}\}} + (MRT - NIQ) \\ &\quad \times \min\{MRT, WIQ\}. \end{aligned} \quad (5)$$



(a) The schedules with mobile devices that have processing ability.

(b) The schedules with mobile devices that do not have processing ability.

Fig. 16. The schedules generated by the scheduling heuristic in Figs. 13 and 14 on scenario with workflows in Fig. 15.

This rule shows that the selection strategy is mainly based on the waiting time of the task (TWT) and the time in the system of the task (TIS). This is because for all the tasks in the waiting queue, the ready time of the processor (MRT), the number of tasks in the waiting queue (NIQ), the work remaining in the waiting queue (WIQ), and the total remaining time in the waiting queue (TTIQ) which are related to the idle processor are all the same and will not be changed by the candidate tasks. Therefore, this sequencing rule can be further simplified as $S_1 \approx TWT$ when $(NIQ + MRT) \geq \frac{TTIQ}{TIS}$, or $S_2 \approx TWT \times TIS$ when $(NIQ + MRT) < \frac{TTIQ}{TIS}$. In other words, this sequencing rule tends to select tasks with smaller waiting time (TWT) and smaller time in the system (TIS).

Overall, based on the above analysis of this scheduling heuristic, we can see that, for the routing decisions, the attributes of the server, such as the bandwidth, play a decisive role in the selection. For the sequencing decisions, the information of tasks, such as the time in the system since the tasks are released, plays a decisive role. This phenomenon is almost consistent with our intuition that processors which are not busy and have short transportation time are good choices to process ready tasks and tasks that stay in the system for a long time should be completed as soon as possible to speed up processing. Based on the feature analysis and this visual presentation of the tree structure, we can identify the importance of different terminals for different rules and allow for a more intuitive understanding of the scheduling heuristics that evolved by the proposed MTGP method.

D. Case Study

In order to give a further understanding of the scheduling process, we give two simple scheduling processes using an evolved scheduling heuristic as shown in Figs. 13 and 14. Two workflows released by two mobile devices are scheduled on a network with two edge servers and two cloud servers. The two workflows have the structure as Fig. 15 and the scheduling results are shown in Fig. 16. Fig. 16(a) gives the scheduling results on the scenario with mobile devices that have processing ability, while Fig. 16(b) shows the scheduling results on the scenario with mobile devices that do not have processing ability.

When the mobile devices have the ability to process tasks, as shown in Fig. 16(a), each mobile device releases a workflow with a DAG structure (workflow 0 is released by mobile device 0, and workflow 1 is released by mobile device 1). In the beginning, there is only one workflow released by mobile device 0, and mobile device 0 decides to upload task 0 of workflow 0 (0.0) to edge 0 to be processed. Then, after the completion of this task, the output data is downloaded from edge 0, and then, the mobile device 0 decides to upload the next ready tasks to edge 0 and edge 1 to be processed, simultaneously. When workflow 1 is released by mobile device 1, mobile device 1 decides to upload all the ready tasks to edge 1 to be processed. As we can see, when there are two workflows in the network, they are not processed one by one but processed at the same time to get a smaller makespan. Finally, after the final task of workflow 1 is completed, the whole work is finished, and we can get the makespan of the process to be about 983 s. Additionally, during the whole process stage, cloud 0 and cloud 1 are not used because of the large upload and download time, while each mobile device is used to process certain tasks. When the mobile devices do not have the ability to process tasks, it can be seen from Fig. 16(b) that, this scheduling heuristics decide to use edge 0 and edge 1 to process most of the tasks and use cloud 1 to process a few tasks. In this case, it takes about 1,174 s to finish the execution of the two workflows, which costs more time.

In conclusion, we can see that the proposed MTGP method can get scheduling heuristics that help mobile devices make decisions. Also, the proposed simulator can emulate the scheduling process in the fog computing environment well. In addition, taking mobile devices as processors can help improve the efficiency of scheduling.

IX. Conclusion

In this article, we present a new DWSFC problem, in which mobile devices have the ability to make autonomous decisions and process tasks, for simulating real-world applications. To solve the new DWSFC problem, the goal of this article is to design a suitable simulator and evolve effective routing rules and sequencing rules simultaneously. This goal has been successfully achieved by the newly proposed simulator and novel MTGP method with newly designed terminals. The proposed simulator is designed as a discrete event-driven simulation process, which simulates the scheduling process by making decisions on two kinds of decision points (i.e., routing decision points and sequencing decision points). The proposed MTGP method can evolve routing rules and sequencing rules simultaneously to meet these decision points. MTGP is examined and compared with four classical scheduling heuristics on nine scenarios. The results suggest that the MTGP method can outperform all the compared approaches in terms of test performance. Additionally, the representative scheduling heuristic gives us a general understanding of which terminals can play a key role. Overall, the newly designed simulator can greatly support studies for the DWSFC problem and the proposed MTGP can obtain effective test performance and potentially good interpretable scheduling heuristics.

In the future, we will combine our approach with other techniques to improve performance, such as the surrogate model and transfer learning techniques. In addition, we will consider applying this MTGP method for solving multi-objective DWSFC problems, such as minimising makespan, load balancing, and budget, simultaneously.

REFERENCES

[1] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proc. IEEE 3rd Workshop Workflows Support Large-Scale Sci.*, 2008, pp. 1–10.

[2] I. Taylor, M. Shields, and I. Wang, "Distributed P2P computing within triana: A galaxy visualization test case," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2003, pp. 1–8.

[3] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb, "A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis," in *Workflows for e-Science*, Berlin, Germany: Springer, 2007, pp. 39–59.

[4] N. Antonopoulos and L. Gillam, *Cloud Computing*, vol. 51. Berlin, Germany: Springer, 2010.

[5] B. Zhang, T. Xiang, H. Zhang, T. Li, S. Zhu, and J. Gu, "Dynamic DNN decomposition for lossless synergistic inference," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, 2021, pp. 13–20.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[7] F. Bonomi, "Connected vehicles, the Internet of Things, and fog computing," in *Proc. 8th ACM Int. Workshop Veh. Inter- Netw.*, Las Vegas, USA, 2011, pp. 13–15.

[8] F. Bonomi, "Cloud and fog computing: Trade-offs and applications," in *Proc. Int. Symp. Comput. Architecture, EON Workshop (ISCA)*, Jun. 2011.

[9] D. Tychalas and H. Karatza, "A scheduling algorithm for a fog computing system with bag-of-tasks jobs: Simulation and performance evaluation," *Simul. Modelling Pract. Theory*, vol. 98, 2020, Art. no. 101982.

[10] X. Q. Pham, N. D. Man, N. D. T. Tri, N. Q. Thai, and E. N. Huh, "A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing," *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 11, pp. 1–16, 2017.

[11] J. Liu et al., "Online multi-workflow scheduling under uncertain task execution time in IaaS clouds," *IEEE Trans. Cloud Comput.*, vol. 9, no. 3, pp. 1180–1194, Third Quarter 2021.

[12] Y. Yu, Y. Feng, H. Ma, A. Chen, and C. Wang, "Achieving flexible scheduling of heterogeneous workflows in cloud through a genetic programming based approach," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 3102–3109.

[13] V. Arabnejad, K. Bubendorfer, and B. Ng, "Dynamic multi-workflow scheduling: A deadline and cost-aware approach for commercial clouds," *Future Gener. Comput. Syst.*, vol. 100, pp. 98–108, 2019.

[14] K. Kofler, I. U. Haq, and E. Schikuta, "A parallel branch and bound algorithm for workflow QoS optimization," in *Proc. IEEE Int. Conf. Parallel Process.*, 2009, pp. 478–485.

[15] S. Mohammadi, L. PourKarimi, and H. Pedram, "Integer linear programming-based multi-objective scheduling for scientific workflows in multi-cloud environments," *J. Supercomputing*, vol. 75, no. 10, pp. 6683–6709, 2019.

[16] Z. Chen et al., "Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2019.

[17] C. Jian, M. Tao, and Y. Wang, "A particle swarm optimisation algorithm for cloud-oriented workflow scheduling based on reliability," *Int. J. Comput. Appl. Technol.*, vol. 50, no. 3/4, pp. 220–225, 2014.

[18] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 110–124, Feb. 2016.

[19] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*. Berlin, Germany: Springer, 2005, pp. 127–164.

[20] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Proc. Australas. Joint Conf. Artif. Intell.*, 2018, pp. 472–484.

[21] H. Fan, H. Xiong, and M. Goh, "Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints," *Comput. Operations Res.*, vol. 134, 2021, Art. no. 105401.

[22] S. Shady, T. Kaihara, N. Fujii, and D. Kokuryo, "Evolving dispatching rules using genetic programming for multi-objective dynamic job shop scheduling with machine breakdowns," *Procedia CIRP*, vol. 104, pp. 411–416, 2021.

[23] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 552–566, Jun. 2021.

[24] E. K. Burke, M. R. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *Parallel Problem Solving From Nature-PPSN IX*. Berlin, Germany: Springer, 2006, pp. 860–869.

[25] K. Sim and E. Hart, "Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model," in *Proc. Conf. Genet. Evol. Comput.*, 2013, pp. 1549–1556.

[26] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evol. Comput.*, vol. 20, no. 1, pp. 63–89, 2012.

[27] T. Weise, A. Devert, and K. Tang, "A developmental solution to (dynamic) capacitated arc routing problems using genetic programming," in *Proc. Conf. Genet. Evol. Comput.*, 2012, pp. 831–838.

[28] M. A. Ardeh, Y. Mei, and M. Zhang, "Genetic programming with knowledge transfer and guided search for uncertain capacitated arc routing problem," *IEEE Trans. Evol. Comput.*, vol. 26, no. 4, pp. 765–779, Aug. 2022.

[29] S. Wang, Y. Mei, and M. Zhang, "A two-stage multi-objective genetic programming with archive for uncertain capacitated arc routing problem," in *Proc. Genet. Evol. Comput. Conf.*, 2021, pp. 287–295.

[30] K.-R. Escott, H. Ma, and G. Chen, "Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2020, pp. 76–90.

[31] Y. Yang, G. Chen, H. Ma, M. Zhang, and V. Huang, "Budget and SLA aware dynamic workflow scheduling in cloud computing with heterogeneous resources," in *Proc. IEEE Congr. Evol. Comput.*, 2021, pp. 2141–2148.

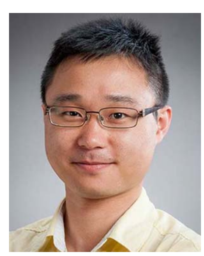
[32] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: A survey," *J. Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[33] A. Caminero, A. Sulistio, B. Caminero, C. Carrión, and R. Buyya, "Extending gridsim with an architecture for failure detection," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2007, pp. 1–8.

[34] L. Zeng, B. Veeravalli, and X. Li, "SABA: A security-aware and budget-aware workflow scheduling strategy in clouds," *J. Parallel Distrib. Comput.*, vol. 75, pp. 141–151, 2015.

[35] E. K. Burke et al., "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.
[36] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statist. Comput.*, vol. 4, no. 2, pp. 87–112, 1994.
[37] S. Nguyen, Y. Mei, B. Xue, and M. Zhang, "A hybrid genetic programming algorithm for automated design of dispatching rules," *Evol. Comput.*, vol. 27, no. 3, pp. 467–496, 2019.
[38] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Trans. Cybern.*, vol. 51, no. 4, pp. 1797–1811, Apr. 2021.
[39] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming-based generative hyperheuristics: A case study in dynamic scheduling," *IEEE Trans. Cybern.*, vol. 52, no. 10, pp. 10515–10528, Oct. 2022.
[40] H. Jabeen et al., "Review of classification using genetic programming," *Int. J. Eng. Sci. Technol.*, vol. 2, no. 2, pp. 94–103, 2010.
[41] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency Comput. Pract. Exp.*, vol. 14, no. 13/15, pp. 1175–1220, 2002.
[42] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.
[43] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," in *Proc. IEEE Int. Conf. E-Sci.*, 2012, pp. 1–8.
[44] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Exp.*, vol. 47, no. 9, pp. 1275–1296, 2017.
[45] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorithm for grid task scheduling," in *Proc. IEEE/IFIP Int. Conf. Central Asia Internet*, 2007, pp. 1–7.
[46] S. Parsa and R. Entezari-Maleki, "RASA: A new grid task scheduling algorithm," *Int. J. Digit. Content Technol. Appl.*, vol. 3, no. 4, pp. 91–99, 2009.
[47] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proc. IEEE Int. Conf. High Perform. Comput. Simul.*, 2009, pp. 1–11.
[48] D. Sirisha and G. Vijayakumari, "Exploring the efficacy of branch and bound strategy for scheduling workflows on heterogeneous computing systems," *Procedia Comput. Sci.*, vol. 93, pp. 315–323, 2016.
[49] H. Tian and J. Chen, "Analysis of overall assignment and sorting of tasks in heterogeneous computing systems based on mathematical programming algorithms," *Wireless Pers. Commun.*, vol. 126, pp. 2283–2301, 2022.
[50] S. K. Misra and P. Kuila, "Energy-efficient task scheduling using quantum-inspired genetic algorithm for cloud data center," in *Proc. Adv. Comput. Paradigms Hybrid Intell. Comput.*, Springer, 2022, pp. 467–477.
[51] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," *IEEE Trans. Serv. Comput.*, vol. 11, no. 4, pp. 727–739, Jul./Aug. 2018.
[52] Z. Chen, K. Du, Z. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, 2015, pp. 708–714.
[53] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2010, pp. 400–407.
[54] W. Chen and J. Zhang, "An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements," *IEEE Trans. Syst., Man, Cyber., C. (Appl. Rev.)*, vol. 39, no. 1, pp. 29–43, Jan. 2009.
[55] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[56] W. Zhao and J. A. Stankovic, "Performance analysis of FCFS and improved FCFS scheduling algorithms for dynamic real-time computer systems," in *Proc. IEEE Real-Time Syst. Symp.*, 1989, pp. 156–157.
[57] J. Blythe et al., "Task scheduling strategies for workflow-based applications in grids," in *Proc. IEEE Int. Symp. Cluster Comput. Grid*, 2005, pp. 759–767.
[58] T. D. Braun et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.
[59] S. Luke and L. Panait, "A survey and comparison of tree generation algorithms," in *Proc. Genet. Evol. Comput. Conf.*, 2001, pp. 81–88.
[60] T. Blickle, "Tournament selection," *Evol. Comput.*, vol. 1, pp. 181–186, 2000.
[61] P. Salot, "A survey of various scheduling algorithm in cloud computing environment," *Int. J. Res. Eng. Technol.*, vol. 2, no. 2, pp. 131–135, 2013.
[62] X. Tang et al., "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 9, pp. 2079–2092, Sep. 2022.
[63] R. Medara and R. S. Singh, "Energy efficient and reliability aware workflow task scheduling in cloud environment," *Wireless Pers. Commun.*, vol. 119, no. 2, pp. 1301–1320, 2021.
[64] B. Dougani and A. Dennai, "Makespan optimization of workflow application based on bandwidth allocation algorithm in fog-cloud environment," PREPRINT (Version 1) available at Research Square, Jul. 6, 2022, doi: 10.21203/rs.3.rs-1809172/v1.
[65] R. Jain, *The Art of Computer Systems Performance Analysis*. Hoboken, NJ, USA: Wiley, 2008.
[66] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," in *Proc. Asia-Pacific Conf. Simulated Evol. Learn.*, 2017, pp. 435–447.

Meng Xu (Student Member, IEEE) received the BSc and MSc degrees from the Bejing Institute of Technology, Beijing, China, in 2017 and 2020, respectively. She is currently working toward the PhD degree in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. Her current research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, and workflow scheduling.



Yi Mei (Senior Member, IEEE) received the BSc and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is an associate professor with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation for combinatorial optimisation, genetic programming, and hyper-heuristic, etc. He has more than 180 fully referred publications, including the top journals in EC such as *IEEE Transactions on Evolutionary Computation*, *IEEE Transactions on Cybernetics*, *Evolutionary Computation Journal*, *European Journal of Operational Research*, *ACM Transactions on Mathematical Software*. He is an associate editor of *IEEE Transactions on Evolutionary Computation*, and a guest editor of the *Genetic Programming Evolvable Machine Journal*.

Shiqiang Zhu received the BSc degree in mechanical engineering from Zhejiang University, Hangzhou, China, in 1988, the MSc degree in mechatronic engineering from the Beijing Institute of Technology, Beijing, China, in 1991, and the PhD degree in mechanical engineering from Zhejiang University, in 1995. He became a faculty with Zhejiang University, in 1995 and was promoted to the rank of professor, in 2001. He is also the director of Zhejiang Laboratory.

Beibei Zhang received the BSc (Hons) degree in information technology from The Hong Kong Polytechnic University, in 2017 and the ME degree from the Department of Electrical and Computer Engineering, University of Toronto, in 2020. He is currently a research engineer in Zhejiang Lab, Hangzhou. His research interests include distributed systems, cloud computing, and peer-to-peer networks.

Tian Xiang received the BSc degree in electrical engineering from Sichuan University, in 2008 and the PhD degree from the College of Information Science & Electronic Engineering, Zhejiang University, in 2013. She is currently a senior researcher with Zhejiang Lab, Hangzhou. Her research interests are in the areas of edge computing, artificial intelligence, and scheduling.

Fangfang Zhang (Member, IEEE) received the BSc and MSc degrees from Shenzhen University, China, and the PhD degree in computer science from the Victoria University of Wellington, New Zealand, in 2014, 2017, and 2021, respectively. She is currently a postdoctoral research fellow in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. She has more than 45 papers in refereed international journals and conferences. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning.

Mengjie Zhang (Fellow, IEEE) received the BE and ME degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, and the PhD degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively. He is a professor of computer science, head of the Evolutionary Computation Research Group. His research interests include evolutionary computation, genetic programming, multi-objective optimization, job shop scheduling. He is a fellow of Royal Society of New Zealand, a fellow of Engineering New Zealand, an IEEE CIS distinguished lecturer.