Multi-Tree Genetic Programming Hyper-Heuristic for Dynamic Flexible Workflow Scheduling in Multi-Clouds

Zaixing Sun, Yi Mei, *Senior Member, IEEE,* Fangfang Zhang, *Member, IEEE,* Hejiao Huang, Chonglin Gu, and Mengjie Zhang, *Fellow, IEEE*

Abstract—Multi-cloud is a promising paradigm due to its advantages such as avoiding vendor lock-in and optimising costs. This paper focuses on dynamic flexible workflow scheduling with minimum total monetary cost in multi-clouds, considering multiple categories of services for each cloud with different configurations and billing methods. Existing studies generally ignore the characteristics and states of each individual cloud when making schedules, which may be ineffective regarding cost savings and quality of service. To address this issue, we propose to introduce a cloud selection decision on top of the existing task selection and resource selection decisions to help us select appropriate resource for task in an overall cost-effective cloud. To automatically learn the task, cloud and resource selection rules simultaneously, we propose a new genetic programming with multi-tree representation based on a customised discrete event-driven dynamic workflow scheduling simulator. Simulation results based on two real-world data traces show that the proposed algorithm performs significantly better than the state-of-the-art algorithms in terms of reducing the rental costs and deadline deviation, and improving the success rate. The results also show that the superiority of the proposed algorithm lies in the ability to select an appropriate cloud resource for a task.

Index Terms—Dynamic workflow scheduling, multi-clouds, genetic programming, hyper-heuristic, deadline constraint.

1 Introduction

Cloud computing, a revolutionary paradigm, not only eliminates hardware investment and maintenance costs for users, but also reduces computing costs by supporting a pay-as-you-go pricing (only pay for what you use) that flexibly provisions resources (e.g., Infrastructure, Software, Function) for users on demand [1], [2]. Multi-Clouds (MCs) have become a popular choice for organisations to achieve greater flexibility, reliability and cost efficiency, which are difficult to achieve with single-cloud setups. The *Flexera 2023 State of the Cloud Report* [3] shows that 87% of the organisations surveyed are using MCs strategies. Cloud service broker provides tenants with a unified and enhanced management interface for MCs and plays a strategic role in helping tenants optimise resource management and deployment across MCs [4].

Data analysis and processing flow applications submitted by users for scientific innovation and knowledge discovery are often structured as Directed Acyclic Graphs (DAGs), called workflows, to represent various tasks with dependent relationships [5], [6]. These workflows are typically deadline-constrained and require timely execution using massive and distributed computing resources. In MCs, the focus is on leveraging the resources provided by multiple Cloud Service Providers (CSPs) to meet various workflows needs rather than dedicating the environment to a single workflow [7]. In addition, organisations typically have multiple workflows running concurrently, each with its own unique characteristics, requirements, and non-periodic arrival (submission) time, i.e. dynamic workflow scheduling. Dynamic workflow scheduling involves complex decision scenarios, bringing new technical challenge mainly in the design of algorithms: Algorithms need to be highly adaptive, able to respond in real-time or near real-time to changes in workflow requirements, cloud resource availability and cost structures. This requires the ability to quickly re-evaluate and modify scheduling decisions without significant overhead. The key to this challenge is to design advanced scheduling mechanisms that are cost effective while maintaining high standards of performance and reliability in an inherently variable environment.

In fact, a CSP can provide a variety of products or services on demand, which ultimately meet personalised requirements by optimising task scheduling and task-to-resource mapping [8], [9]. These service categories can be Infrastructure-as-a-Service (IaaS, e.g., AWS EC2), Platform-as-a-Service (PaaS, e.g., Google App Engine), Software-as-a-Service (SaaS, e.g., Office365), and Function-as-a-Service (FaaS, e.g., AWS Lambda), etc. Enterprises using a MCs with multiple categories of services can increase their flexibility and choice in selecting the suitable platform for specific

*Manuscript received *; revised *; accepted *. This work is supported by the Shenzhen Science and Technology Program under Grant No. GXWD20220817124827001, and No. JCYJ20210324132406016. (Corresponding author: Chonglin Gu.)*

Z. Sun, H. Huang and C. Gu, are with the School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen 518000, China, and also with Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen 518000, China. Z. Sun is also with the Evolutionary Computation Research Group, Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand. E-mail: szx_1010@stu.hit.edu.cn, {huanghejiao, guchonglin}@hit.edu.cn.
Y. Mei, F. Zhang, and M. Zhang are with the Evolutionary Computation Research Group, Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand. E-mail: {yi.mei, fangfang.zhang, mengjie.zhang}@ecs.vuw.ac.nz.

tasks; reduce dependence on a single provider; enhance flexibility through redundancy and failover options; and promote innovation and acquire the latest technology to improve core competitiveness [10], [11]. Based on IaaS and PaaS, Sharma et al. [12] created a mixed-integer non-linear programming model for variable cloud components for small application developer firms. The model rationalises the allocation of the applications on IaaS and PaaS clouds by considering factors such as computing power hours, storage requirements, and total cost. Most existing studies focus on a single service category such as Virtual Machines (VMs) [13], [14] and functions [11], [15], [16]. In addition, for heterogeneous tasks, there are privacy-sensitive [13], [14], [17], computation-intensive or memory-intensive [18], [19] in the existing studies. Since CSPs can offer multiple categories of services, we name the dynamic workflow scheduling in the MCs as *the dynamic flexible workflow scheduling problem*. The workflow's flexibility lies in its tasks being associated with specific service categories.

In this paper, we focus on the dynamic flexible workflows scheduling under deadline constraints in multi-clouds with multiple service categories (DFWS-MCs). The objective is to minimise the total monetary cost, which includes the execution cost of tasks and the communication cost when communicating between different CSPs. It is worth mentioning that Cisco has designed Software-Defined Wide Area Network to address the network bandwidth challenge, and thus, the network bandwidth is not necessarily to be a bottleneck in multi-clouds anymore. Previous works [20], [21] have studied the dynamic workflow scheduling in multi-clouds, but only consider VM services. Therefore, in terms of the problem model, we differ from existing researches in that we consider multi-cloud environments with multiple service categories. DFWS-MCs inherits many challenges of workflow scheduling in traditional cloud environments, e.g., deadline constraints; determining the scheduling sequence of tasks; billing for cross-cloud data transmission; capacity and cost trade-offs in resource selection [15], [22], [23]. Moreover, the solution space becomes larger in MCs, making these challenges more difficult to overcome. Since we consider multiple service categories, this leads to the following extra challenges in DFWS-MCs.

First, different service categories have different configuration parameters, and their billing methods may be significantly different. For example, the VM service' billing method is only the execution time multiplied by the unit price, AWS lambda functions' include a per GB second charge and a per invocation charge, and Google functions are billed differently than AWS lambda functions [24], [25]. Moreover, due to different configurations, we should make decisions according to different service categories when solving this problem. We also need to consider other factors, such as the longer cold booting time of VM services than that of function services and the maximal time per execution of the function instances.

Second, traditional workflow scheduling in cloud usually involves two decisions [6], [11], [21], [26]: the task prioritisation and the resource selection. Existing studies on MCs always select resources from all available resources, and then the cloud owning the selected resource is automatically determined. Although this way is relatively naive, it is difficult to select the most appropriate resource for the workflows because the number of available heterogeneous resources is usually very large and it will be varying constantly during the scheduling process. Moreover, this method also ignores the characteristics and states of each individual cloud when making dynamic schedules, which may be ineffective in selecting an appropriate resource to execute workflows with minimum cost under deadline constraints. For example, selecting low-configuration resources with low cost to execute tasks may miss the deadline, while selecting high-configuration takes less execution time but with high expenditure. Therefore, we propose to comprehensively evaluate the priority of different clouds based on the system state of each cloud and the known task allocation. We will subsequently select resources within the belonged cloud with the top priority. This is expected not only to choose an overall cost-effective cloud for tasks, so as to select resources that are suitable in terms of time and cost, but also reduce the selection space of the resources.

At present, many studies [7], [21], [27] employ heuristic algorithms to solve dynamic (real-time or online) workflow scheduling. These algorithms are usually based on extracting state features during the scheduling process to manually design rules, which highly relies on domain knowledge. In addition, human experts are often unable to identify all the subtle and interrelated conditions between the different types of features to create and evaluate rules [28], [29]. The decisions made by manually designed scheduling heuristics are mostly greedy decisions, which might not be good for long-term scheduling [30]. *Hyper-heuristic* [31] is an automated methodology for selecting or generating heuristics to solve hard computational problems, thereby improving the ability to adapt and capture important factors. The goal of this approach is to explore the "heuristic search space" of the problems instead of the solution search space in the cases of heuristics and meta-heuristics. Different from genetic algorithms, Genetic Programming Hyper-Heuristic (GPHH) [32], [33] is to automatically evolve scheduling heuristic rules or algorithms rather than finding the solution directly. In other words, the output of GP or GPHH is a set of rules or heuristics/policies instead of a schedule or a solution, which is typically the case for GA or other heuristic methods. Being a hyper-heuristic approach, GPHH consists of an *offline training phase* and an *online test phase*. During the offline training phase, GPHH is designed to evolve a scheduling heuristic, driven by the goal to minimise the total monetary cost when executing a group of workflows. The best heuristic discovered by GPHH during training can make decisions in dynamic/real-time workflow scheduling. GPHH has been successful for automatically evolving scheduling heuristics for many combinatorial optimisation problems, such as job shop scheduling [34], [35] and routing problems [36], [37]. However, the existing GPHH methods for cloud workflow scheduling studies [6], [30], [38], [39] support up to two rules that cannot be directly used for our three decision-making problems. To address this challenge, this paper proposes a Multi-Tree Genetic Programming (MTGP) to evolve scheduling heuristics for DFWS-MCs. The main contributions of this paper are shown as follows:

- We provide a constrained optimisation model for the dynamic flexible workflow scheduling under deadline constraints in multi-clouds with multiple service categories, which covers the important aspect of multiple service categories not yet considered in the literature.

- We customise a new dynamic workflow scheduling simulator to mimic the scheduling process in real-world scenarios as the existing simulator does not support multiple service categories. The scheduling heuristic is used to make a decision at each decision point to obtain the final schedules.
- We propose a MTGP method with a new representation, which is comprised of three rules, i.e., the *task selection rule* to select a ready task, the *cloud selection rule* to select a cloud to execute the ready task, and the *resource selection rule* to select a resource in the selected cloud to execute the ready task. We design a number of terminals based on the scheduling process, which can clearly describe the state of the multi-cloud system. The designed terminals are divided into three different groups to construct the above three rules.
- We conduct extensive simulation experiments based on two real-world data traces (the workflows extracted from Cluster-trace-v2018 of the Alibaba Cloud and scientific workflows from Pegasus project). Experimental results show that the proposed algorithm outperforms existing state-of-the-art methods. The results also show that the proposed algorithm learns and uses more valuable terminals to select cloud and resource to execute task from a holistic perspective.

2 Related Work

2.1 Workflow Scheduling in Different Clouds

Early research in the heterogeneous distributed computing environment, as an embryonic form of cloud computing, focuses mainly on the effective management and utilisation of computing resources. The widely recognised and popular HEFT algorithm [40] was originated in such an environment. In the heterogeneous environment, the estimated execution time of tasks on processors often remained independent of processor performance [40], [41], whereas in a cloud environment it is positively correlated with processor performance. Accordingly, task scheduling in the heterogeneous environment can be analogised to classical flexible job shop scheduling, where the candidate machines for each task are all processors.

Besides minimising makespan under deadline constraints, the Workflow Scheduling Problem (WSP) has different additional concerns in different cloud environments. In traditional or private clouds, WSP usually focuses on load balancing and energy consumption. Based on queue balancing and data reuse-replication techniques, Casas et al. [42] developed a load balancing scheduler for scientific workflows scheduling in cloud to simultaneously minimise execution time and monetary cost. Xia et al. [43] combined genetic algorithm with longest common subsequence to optimise the makespan and energy consumption for workflow scheduling in cloud. In public cloud or multi-cloud, WSP usually focuses on issues related to minimising the user's leasing costs used to execute the tasks or maximising the provider's interests. Sun et al. [22] proposed an enhanced task type first algorithm to minimise the total cost and total idle rate for deadline-constrained workflow scheduling in public cloud. They considered some new features of cloud environments such as hibernation and per second billing. Taghinezhad-Niar and Taheri [21] proposed a Reliability, rental cost, and Energy-aware Multi-workflow Scheduling in the Multi-cloud (REMSM) heuristic algorithm and a DVFS-enabled version of it for multi-cloud systems, with the aim of minimising monetary cost and energy consumption while maximising application execution reliability. Cong et al. [44] developed a personalised service request model and a user satisfaction prediction model for a cloud service platform. Then, based on these models, they proposed a scheduling scheme based on lightweight value assessment and cross-entropy to maximise the cloud service provider's profit. However, these studies are limited to manually designed heuristics, which require empirical knowledge and ingenious design to achieve good results.

In hybrid clouds (private cloud+public clouds), WSP primarily ensures the privacy or security of tasks and then tackles environment-specific issues. Sun et al. [14] proposed a nested algorithm based on multi-objective salp swarm algorithm and iterative greedy algorithm for hybrid-cloud-based privacy-aware multi-workflow scheduling. They considered the trade-offs between the three minimisation objectives workflow-oriented total tardiness, private-cloud-oriented total energy consumption, and public-cloud-oriented total monetary cost. In fog computing or edge computing environment, WSP involves efficiently allocating tasks to available resources within fog or edge nodes and ensuring short response time and high reliability [45]. Aburukba et al. [46] developed an integer linear programming model for the IoT requests scheduling problem in hybrid fog-cloud computing. These requests were generated by the edge devices and need to be allocated to the available resources in the cloud and fog with lower latency, where the latency can be due to transmission delay, routing or queuing delay, and waiting time if the resources are busy, and other components. In recent years, the emergence of cloud native technologies, such as microservices and containers, as well as serverless or FaaS platforms, has introduced new considerations for WSP, which impact the design, deployment, and scheduling of workflows. Ranjan et al. [47] developed a container-based virtualised model for energy-efficient workflow scheduling in software-defined data centers. Since containers can be faster to manage and deploy than VMs, which can save time and energy consumption with application deployment, they focused on migrating containers within VMs among different data centers. Based on FaaS platforms, users mainly focus on the lifecycle of the function without considering the infrastructure, but still need to declare the appropriate function configuration to meet the deadline and budget requirements of the function [48]. Although current cloud environments and their collaboration environments are complex, diverse and have different functions, existing research is limited to supporting a single category of task execution, but cannot handle multiple categories.

To address the above issues, this paper studies a new dynamic workflow scheduling in MCs, which supports multiple categories of task execution. To this end, we propose a learning and hyper-heuristic algorithm (i.e., GPHH) to automatically design and evolve heuristics for the problem.

2.2 GPHH for Cloud Workflow Scheduling

Heuristic algorithms (designed based on expert experience) have been extensively developed and employed to solve workflow scheduling problems [49]. GPHH searches for a heuristic that can then be applied to construct corresponding schedules to different problem instances, rather than searching directly for solutions to a particular problem instance [50]. Many studies have shown that production scheduling heuristics learned by GPHH can outperform manually designed *heuristics* in the literature [32], [50], especially in dynamic and stochastic environments. However, the studies of GPHH in the field of cloud computing are limited. Escott et al. [38] used GPHH to minimise the overall makespan for dynamic workflow scheduling. Yang et al. [39] employed GPHH to minimise the VM rental fees and SLA penalties for the dynamic WSP. They extracted some new factors as terminals to help GPHH get better performance. In [38], [39], each individual of GPHH has one tree which is used to select a processor for each task. Xiao et al. [6] proposed a cooperative coevolution GPHH to minimise the makespan for static WSP, which co-evolves the task selection rule and the resource selection rule simultaneously. Xu et al. [30] established a new model for dynamic WSP in fog computing which considered the limited computing resources (e.g., mobile devices, edge nodes and cloud servers) in the real world. They developed a discrete event-driven simulator and proposed a GPHH to minimise the overall makespan, which also evolves the task and resource selection rules simultaneously. However, the terminals used by the existing studies to construct rules are naive and limited, which cannot fully reflect the state of the cloud system. Moreover, their GPHH methods contain only two rules, thus cannot be directly used for our three decision-making problems.

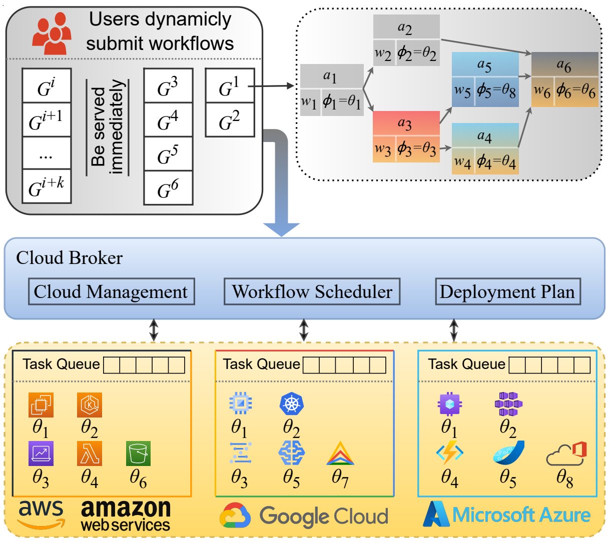
The HSA9Fs algorithm proposed by Sun et al. [14] based on the extracted 9 features outperforms existing state-of-the-art algorithms in some cases, showing that extracting more factors can help find more effective rules. Therefore, we consider designing more features that can reflect the system state to construct three different rules.

3 Cloud Workflow Scheduling Model

3.1 Multi-clouds Architecture

This section introduces the dynamic workflow scheduling architecture in multi-clouds with multiple service categories, as shown in Fig. 1. The architecture consists of the following three modules:

- **Multi-cloud environment**. In multi-cloud environment, there are multiple CSPs, each of which provides different categories and prices of service instances. Different types of service instances may have different application scenarios, e.g., θ_1 is a VM instance type, θ_2 is an analysis service, θ_3 is a container service, and θ_5 can be directly used for machine learning.
- **Users dynamically submit workflows**. The information of a workflow is unknown until it is submitted, and multiple workflows can be submitted simultaneously. CSPs can provide an unlimited number of different service instances to the end users [7], [22] . Therefore, each workflow can be served immediately when submitted.



Users dynamicly submit workflows
G^i
G^{i+1}
...
G^{i+k}
Be served immediately
G^3
G^4
G^5
G^6
G^1
G^2
a_1 w_1 $\phi_1=\theta_1$
a_2 w_2 $\phi_2=\theta_2$
a_3 w_3 $\phi_3=\theta_3$
a_4 w_4 $\phi_4=\theta_4$
a_5 w_5 $\phi_5=\theta_8$
a_6 w_6 $\phi_6=\theta_6$
Cloud Broker
Cloud Management
Workflow Scheduler
Deployment Plan
Task Queue
θ_1 θ_2 θ_3 θ_4 θ_6
aws amazon web services
θ_1 θ_2 θ_3 θ_5 θ_7
Google Cloud
θ_1 θ_2 θ_4 θ_5 θ_8
Microsoft Azure

Fig. 1. The dynamic workflow scheduling architecture in multi-clouds with multiple service categories.

Table 1
Symbols and Meanings.

Symbol	Definition
$\mathcal{P}$	The set of CSPs.
Θ	The set of all the service categories provided by the CSPs.
$\mathcal{S}(p)$	The set of service instance types provided by the CSP $p \in \mathcal{P}$.
$\theta(s)$	The category associated with instance type s, $\theta(s) \in \Theta$.
$v(s), m(s), q(s)$	The number of vCPUs, memory size, and computational capacity associated with instance type s.
$c(s), b(s)$	The unit price and billing interval associated with instance type s.
$W(p_1, p_2)$	The bandwidth between the CSPs p_1 and p_2 for data transfer, where $W(p,p) = W(p)$ is the interval bandwidth of the CSP $p \in \mathcal{P}$.
$\mathcal{G}$	A set of workflow applications.
$T(g)$	A set of tasks in workflow $g \in \mathcal{G}$.
$s(t)$	The computation size associated with task $t \in T(g)$.
$n(t)$	The number of invocations associated with task t.
$T^{pred}(t)$	A set of immediate predecessors of task t, $T^{pred}(t) \subseteq T(g)$.
$T^{succ}(t)$	A set of immediate successors of task t, $T^{succ}(t) \subseteq T(g)$.
$d(t_1, t_2)$	The data required to be transferred from each task t_1 to its successor task $t_2 \in T^{succ}(t)$.
$\phi(t)$	The service category required to execute a task t.
$a(g)$	The arrival time of the workflow g.
$\rho(g)$	The deadline of the workflow g.
$x(t)$	The execution start time of the task t in the schedule.
$y(t)$	The assigned instance of the task t in the schedule.
$z(t,p)$	A boolean variable indicating whether $y(t)$ belongs to the CSP p or not.
$\tau^e(t)$	The execution time of task t on $y(t)$.
$\tau^c(t_1, t_2)$	The communication time between each task t_1 and its successor task t_2.
$\mathbb{I}_{y(t_1) \neq y(t_2)}$	A boolean variable, which is equal to 1 if $y(t_1) \neq y(t_2)$, and 0 otherwise.
$\pi^e(t, \tau^e(t), \phi(t))$	The execution cost of the task t.
$\pi^c(t_1, t_2)$	The communication cost between a task t_1 and its successor task $t_2 \in T^{succ}(t_1)$.

- **Cloud broker**. The two main functions of the cloud broker are [11]: a) optimising the configuration of virtual resources across a group of CSPs, and b) monitoring and managing these virtual resources. The cloud broker's scheduler component aims to generate a near-optimal deployment plan based on user's demands on resources and cloud service offerings provided by CSPs.

3.2 Problem Formulation

Let $\mathcal{P} = \{AWS, GoogleCloud, Azure, \cdots\}$ be the set of CSPs, and $\Theta = \{VM, AWSLambda, GoogleFunction, \cdots\}$ be the set of all the service categories provided by the

CSPs. $\mathcal{S}(p)$ is the set of service instance types provided by the CSP $p \in \mathcal{P}$. Each instance type s is associated with a category $\theta(s) \in \Theta$, the number of vCPUs $v(s)$, required memory $m(s)$, computational capacity $q(s)$, unit price $c(s)$, and billing interval $b(s)$. Let $W(p_1, p_2)$ be the bandwidth between the CSPs p_1 and p_2 for data transfer, where $W(p,p) = W(p)$ is the interval bandwidth of the CSP $p \in \mathcal{P}$.

Customers dynamically submit their applications to the multi-cloud platform. Before being submitted, the features of customers' applications (e.g. arrival time, deadlines, and structures, etc.) are unknown to the platform. Let $\mathcal{G}$ be a set of dynamic workflow applications. Each workflow $g \in \mathcal{G}$ contains a set of tasks $T(g)$. Each task $t \in T(g)$ is associated with a computation size $s(t)$, number of invocations $n(t)$, a set of predecessor tasks $T^{pred}(t) \subseteq T(g)$, and a set of successor tasks $T^{succ}(t) \subseteq T(g)$, representing the data dependencies between tasks. The data required to be transferred from each task t_1 to its successor task $t_2 \in T^{succ}(t)$ is denoted as $d(t_1, t_2)$. Each task t requires the service category $\phi(t)$ to execute it, which $\phi(t) \in \Theta$. The arrival time of the workflow g is $a(g)$, and its deadline is $\rho(g)$.

The pricing scheme used by each service category may vary. For example, VM-based service categories may be billed based on lease time, while function-based service categories may be billed based on the number of invocations. In addition, major CSPs now support per-second billing, which brings customers closer to being billed only for the time they actually occupy resources. For the sake of simplicity, we denote the execution cost of a task t as $\pi^e(t, \tau^e(t), \phi(t))$, where $\tau^e(t)$ is the execution time of t. For communication, the price of transferring a unit of data between CSPs p_1 and p_2 is $\pi^c(p_1, p_2)$, where $\pi^c(p_1, p_2) = 0$ if $p_1 = p_2$.

The problem aims to assign resource instances from CSPs to execute the tasks of the workflows subject to the following constraints:

- No task can be executed until all its predecessor tasks have been completed.
- Each task must be executed by an instance with the category it requires.

Let $x(t)$ and $y(t)$ be the execution start time and assigned instance of the task t in the schedule, and $z(t,p)$ be an auxiliary variable indicating whether $y(t)$ belongs to the CSP p or not, i.e.,

$$z(t,p) = \begin{cases} 1, & \text{if } y(t) \text{ belongs to } p, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The execution time of task t on $y(t)$ is calculated as

$$\tau^e(t) = \frac{s(t)}{q(y(t))}. \tag{2}$$

The communication time between each task t_1 and its successor task t_2 is

$$\tau^c(t_1, t_2) = \sum_{p_1 \in \mathcal{P}} \sum_{p_2 \in \mathcal{P}} z(t_1, p_1) z(t_2, p_2) \frac{d(t_1, t_2)}{W(p_1, p_2)} \mathbb{I}_{y(t_1) \neq y(t_2)}, \tag{3}$$

where $\mathbb{I}_{y(t_1) \neq y(t_2)} = 1$ if $y(t_1) \neq y(t_2)$, and 0 otherwise. When $y(t_1) \neq y(t_2)$, it means that two tasks are not allocated the same resource, and they require data transmission.

The total monetary cost includes the execution cost of tasks and the communication cost when communicating between different CSPs. The execution cost is given by $\pi^e(t, \tau^e(t), \phi(t))$. Since there are different billing methods for different categories of resources or different providers, we list the following three methods based on the above configuration[1].

- If the service category for executing task t is VM, i.e. $\phi(t) = VM$, the execution cost is the execution time multiplied by the unit price, as shown in Eq. (4).

$$\pi^e(t, \tau^e(t), \phi(t)) = \left\lceil \frac{\tau^e(t)}{b(y(t))} \right\rceil \times \frac{c(y(t))}{3600}. \tag{4}$$

- If the service category for executing task t is a lambda function provided by AWS, i.e. $\phi(t) = AWSLambda$, the execution cost includes a per GB second charge (defined as CPU time multiplied by the amount of memory used.) and a per invocation charge ($0.2 per million requests), as shown in Eq. (5).

$$\pi^e(t, \tau^e(t), \phi(t)) = \left\lceil \frac{\tau^e(t)}{b(y(t))} \right\rceil \times c(y(t)) \times n(t) \times \frac{m(y(t))}{1024} + \left\lceil \frac{n(t)}{10^6} \right\rceil \times 0.2. \tag{5}$$

- If the service category for executing task t is a function provided by Google, i.e. $\phi(t) = GoogleFunction$, the execution cost includes a per GB-second charge, a per GHz-second ($0.01 per thousand GHz-seconds) and a per invocation charge ($0.4 per million requests), as shown in Eq. (6).

$$\pi^e(t, \tau^e(t), \phi(t)) = \left\lceil \frac{\tau^e(t)}{b(y(t))} \right\rceil \times n(t) \times \Big(\frac{m(y(t))}{1024} \times c(y(t)) + \frac{y(t)^{GHz}}{10^6} \times 0.01 \Big) + \left\lceil \frac{n(t)}{10^6} \right\rceil \times 0.4, \tag{6}$$

where $y(t)^{GHz}$ is the GHz configuration of the instance $y(t)$ that executes task t.

The communication cost between a task t_1 and its successor task $t_2 \in T^{succ}(t_1)$ is

$$\pi^c(t_1, t_2) = \sum_{p_1 \in \mathcal{P}} \sum_{p_2 \in \mathcal{P}} z(t_1, p_1) z(t_2, p_2) d(t_1, t_2) \pi^c(p_1, p_2). \tag{7}$$

The problem can then be formulated as

$$\min \sum_{g \in \mathcal{G}} \sum_{t \in T(g)} \Big(\pi^e(t, \tau^e(t), \phi(t)) + \sum_{t' \in T^{succ}(t)} \pi^c(t, t') \Big), \tag{8}$$

$$s.t.: \; x(t) \geq a(g), \; \forall\, g \in \mathcal{G}, \; t \in T(g), \tag{9}$$

$$x(t) + \tau^e(t) \leq \rho(g), \; \forall\, g \in \mathcal{G}, \; t \in T(g), \tag{10}$$

$$x(t_1) + \tau^e(t_1) + \tau^c(t_1, t_2) \leq x(t_2), \forall\, g \in \mathcal{G}, \; t_1 \in T(g), \; t_2 \in T^{succ}(t_1), \tag{11}$$

$$\phi(t) = \theta(y(t)), \; \forall\, g \in \mathcal{G}, \; t \in T(g), \tag{12}$$

$$\sum_{t \in T(g)} \sum_{p \in \mathcal{P}} z(t,p) = 1, \; \forall\, g \in \mathcal{G}. \tag{13}$$

1. We don't consider the free tier of function categories. Moreover, for tasks with multiple invocations of function categories, the algorithm only needs to make a decision once and then execute the task on the same type of function instance.

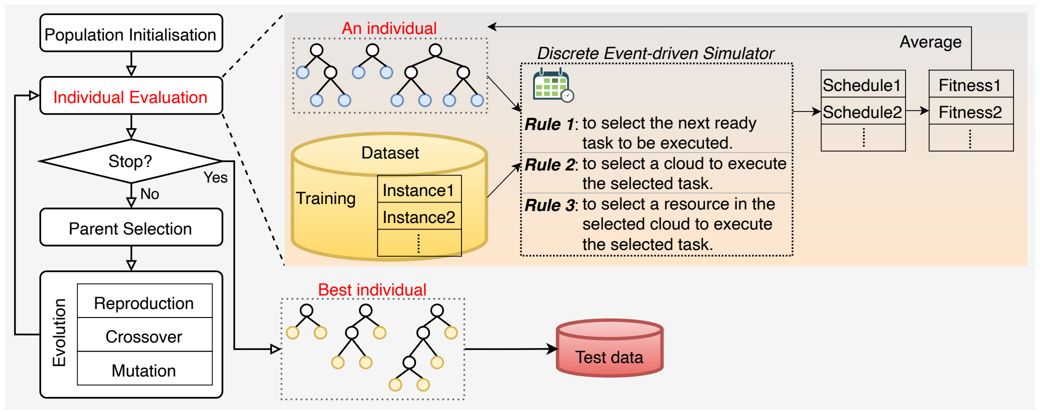

Population Initialisation
Individual Evaluation
Stop?
Yes
No
Parent Selection
Evolution
Reproduction
Crossover
Mutation
An individual
Dataset
Training
Instance1
Instance2
Discrete Event-driven Simulator
Rule 1: to select the next ready task to be executed.
Rule 2: to select a cloud to execute the selected task.
Rule 3: to select a resource in the selected cloud to execute the selected task.
Average
Schedule1
Schedule2
Fitness1
Fitness2
Best individual
Test data

Fig. 2. The flowchart of the MTGP approach. The input to the flowchart is training data. The output are the learned three rules represented by the best individuals.

Eq. (8) is the objective, which is the total monetary cost to be minimised. Constraints (9)-(13) are the constraints to be satisfied by the deployment of tasks after workflow submission. Constraint (9) ensures that each task is executed after submission. At any time, the information about all tasks that have not yet been submitted at that time is not known to the model. Constraint (10) ensures that the workflow is finished before its deadline, which is considered as a soft constraint in this paper and we will evaluate the quality of the solution by the success rate metric. Constraint (11) is the precedence constraint in a workflow, which indicates that the task cannot be executed until all of its immediate predecessors have finished and it has received all of the output data from its immediate predecessors. Constraint (12) implies that the service category required by the task matches the service category that executes it. Constraint (13) means that each task in each workflow is scheduled only once.

4 The Proposed Dynamic Workflow Scheduling Algorithm

This section presents the proposed MTGP approach for the DFWS-MCs. We first give an overview of MTGP, then introduce the representation of the allocation rules, the terminal set, and the fitness function.

4.1 Overview

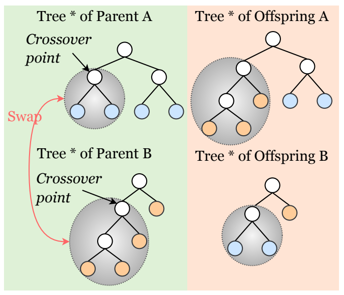
MTGP aims to evolve scheduling heuristics to help the *Workflow Scheduler* to schedule cloud resources for workflow execution. We rely on the dynamic workflow scheduling simulator to simulate the processing of dynamically arriving workflows over an extensive period of time. The details of this simulator are given in Section 4.3 below.

We propose the MTGP approach to automatically generate the following 3 rules to make decisions:

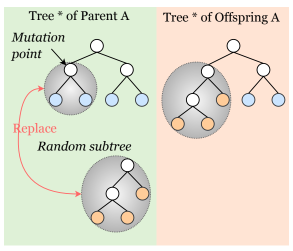
- **Task selection rule** (Rule 1): to select the next ready task to be executed. If there are multiple tasks ready at the same time, the rule is used to determine the sequence of tasks.
- **Cloud selection rule** (Rule 2): to select a cloud to execute the selected task.
- **Instance selection rule** (Rule 3): to select a resource in the selected cloud to execute the selected task. First, it filters out the instances that cannot meet the sub-deadline to execute the task. If no instance remains, then the instance in the cloud with the earliest finish time is selected to execute the task. Otherwise, the instance selection rule is used to select the instance. If there are multiple instances with the same priority, it is preferred to select an existing instance that does not need to be created.

Fig. 2 shows the flowchart of using MTGP to learn heuristics for DFWS-MCs [30], [51]. Each GPHH individual consists of three trees (corresponding to the above three rules). The fitness of an individual depends on the co-operation of the three rules. The details of individual representation are given in Section 4.2. The evolutionary mechanisms of MTGP are presented separately as follows.

- **Population 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 [50]. 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.
- **Individual evaluation**. Given a training instance, we can evaluate the fitness of an individual (i.e., a combination of the three rules) by applying it to the *Workflow Scheduler* or dynamic workflow scheduling simulator. After fitness evaluation, each individual has a fitness that represents its quality.
- **Parent selection**. If the stopping criterion is met, the best individual so far is considered as the best evolved scheduling heuristic for DFWS-MCs. Otherwise, we use the tournament selection to select parents for generating offspring.
- **Evolution**. Evolution has three genetic operators, which are reproduction, crossover and mutation. These operators aim at generating a new population by inheriting good materials from the parent population. For ***reproduction***, we copy a certain number of elites into the next generation directly, where elites are the top individuals picked up from the current population. For ***crossover***, we randomly select a tree with the same decision from two adjacent parents, then randomly select a sub-tree from each parent and swap these two sub-trees, and then randomly select and swap another whole tree with the same decision to produce two new individuals. For ***mutation***, we first randomly select a tree and generate a



(a) The process of crossover.



(b) The process of mutation.

Fig. 3. The processes of crossover and mutation. Only trees with the same function can do crossover and mutation.

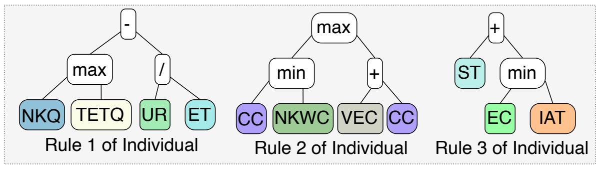


Fig. 4. An example of a tree-based individual representation. The rules can be regarded as priority functions to prioritise tasks/clouds/instances. In this paper, a lower priority value gives the task/cloud/instance a higher precedence. $R1 = \max\{\text{NKQ}, \text{TETQ}\} - \frac{\text{UR}}{\text{ET}}$.
$R2 = \max\{\min\{\text{CC}, \text{NKWC}\}, \text{VEC} + \text{CC}\}$.
$R3 = \text{ST} + \min\{\text{EC}, \text{IAT}\}$.

subtree with the same decision as it. Then we randomly select a subtree from the parent and replace it with the newly generated subtree. The processes of crossover and mutation are shown in Figs. 3(a) and 3(b).

4.2 Representation, Terminal Set, and Function Set

To evolve a scheduling heuristic with three rules for the DFWS-MCs problem, an individual is designed with three trees which represent task selection rule, cloud selection rule and instance selection rule, respectively. The structure of the individual can be seen in Fig. 2, and we also give an example of a tree-based individual in Fig 4. We design 22 terminals based on the scheduling process, which indicate the characteristics related to workflows, tasks, instances and clouds, to describe the state of the multi-cloud system. These terminals, i.e. the low-level heuristics, are designed based on the global real-time information of the workflows and the multi-cloud system, so that the learned heuristics can solve the dynamic workflow scheduling.

Table 2 shows the terminal and function sets of MTGP. The terminals are the leaves of the tree while the functions cannot be located at the leaves of the trees. Note that the terminal set is different for each rule, but the function set is the same. We use the symbol "✓" to mark the terminators that compose the corresponding rule in Table 2. Nos. 1-6 are task-related terminals, Nos. 7-10 are workflow-related terminals, Nos. 11-16 are instance-related terminals and Nos. 17-22 are cloud-related terminals. The **function set** is {+, -, *, protected /, Max, Min}, and each function has two arguments. The protected division "/" returns one if divided by zero. Below we provide additional explanations for those terminals in the table that are not self-explained.

No.4 SD is the sub-deadline of a task, which is set based on the latest finish time, as shown in Eq. (15). Considering that different clouds may have uneven performance, we use the terminal ET (denote as $ET(t)$ for a task t) and the maximum intra-cloud transmission time $\bar{\tau}^c(t_1, t_2) = \frac{d(t_1,t_2)}{\min_{p \in P}\{W(p)\}}$ to estimate the latest finish time $\ell(t)$ (Eq. (14)), rather than the minimum execution time [7], [21] or maximum execution time [14], [16] of tasks. ε is a random factor within the range $[0.95, 1]$. The smaller the ε, the more urgent the tasks in a workflow and the smaller the sub-deadline. Following [14], we set $\varepsilon = 0.95$.

Table 2
Terminal and Function sets of MTGP.

No.	Notation	Description	3-Decision MTGP Rule1	Rule2	Rule3
1	ET	The average execution time of a task in the multi-cloud system.	✓		
2	NPK	The number of direct predecessors for a task.	✓		
3	NSK	The number of direct successors for a task.	✓		
4	SD	The sub-deadline of a task.	✓		
5	UR	The upward rank of a task.	✓		
6	VCT	The average communication time for a task.	✓		
7	NKQ	The number of tasks in ready queue.	✓		
8	NRK	The number of remaining (or unscheduled) tasks in a workflow.	✓		
9	TETQ	The total computation size of tasks in ready queue.	✓		
10	TETRK	The total computation size of remaining tasks in a workflow.	✓		
11	AAT	The actual available time of a instance.			✓
12	AET	The actual execution time of task on an instance.			✓
13	EC	The execution cost of task on an instance.			✓
14	IAT	The available time of an instance for a task.			✓
15	ST	The slack time of task, which is $(AAT-SD)$.			✓
16	CC	The communication cost of a task while it is executed on an instance.		✓	✓
17	NKWC	The number of the scheduled tasks belonging to the same workflow in a cloud.		✓	
18	VAAT	The average AAT in a cloud.		✓	
19	VEC	The average EC in a cloud.		✓	
20	VET	The average ET in a cloud.		✓	
21	VIAT	The average IAT in a cloud.		✓	
22	VST	The average ST in a cloud.		✓	
Function set		+, -, *, protected /, Max, Min	✓	✓	✓

$$\ell(t_1) = \begin{cases} \rho(g), & \text{if } T^{succ}(t_1) = \varnothing, \\ \max\limits_{t_2 \in T^{succ}(t_1)} \{\ell(t_2) - ET(t_2) - \bar{\tau}^c(t_1, t_2)\}, & \text{otherwise.} \end{cases} \tag{14}$$

$$SD(t) = \begin{cases} \ell(t), \text{ if } T^{succ}(t) = \varnothing, \\ \varepsilon \times \ell(t), \quad \text{otherwise.} \end{cases} \tag{15}$$

No.5 UR is the upward rank of a task t, which is the length of the critical path from t to the exit task [40], as shown in Eq. (16).

$$UR(t_1) = \begin{cases} ET(t_1), & \text{if } T^{succ}(t_1) = \varnothing, \\ ET(t_1) + \max\limits_{t_2 \in T^{succ}(t_1)} \{UR(t_2) + \bar{\tau}^c(t_1, t_2)\}, & \text{otherwise.} \end{cases} \tag{16}$$

No.11 AAT is the actual available time of an instance for a task and is equal to the maximum of the current system time, IAT and data transfer finish time from the predecessor tasks. **No.13** EC is the execution cost of a task on an instance. If the instance is newly created, then the EC will include the cost incurred for booting the instance. **No.14** IAT is the available time of an instance, which is the finish time of the last task assigned to the instance. If the instance is newly created, IAT is set to the system time plus the booting time of the instance. **No.16** CC is the communication cost of a

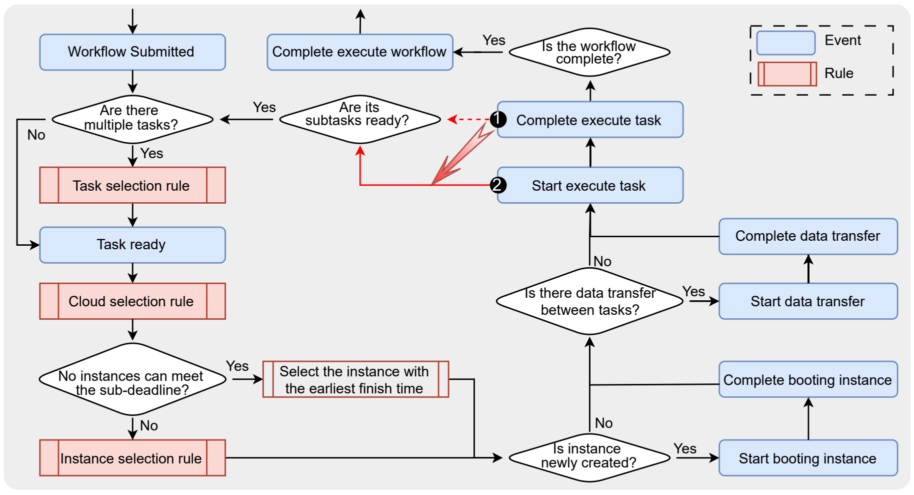

Workflow Submitted
Complete execute workflow
Yes
Is the workflow complete?
Event
Rule
Are there multiple tasks?
Yes
Are its subtasks ready?
Complete execute task
No
Yes
Task selection rule
Start execute task
Task ready
Complete data transfer
No
Cloud selection rule
Is there data transfer between tasks?
Yes
Start data transfer
No instances can meet the sub-deadline?
Yes
Select the instance with the earliest finish time
Complete booting instance
No
No
Instance selection rule
Is instance newly created?
Yes
Start booting instance

Fig. 5. The flowchart of dynamic event triggering when scheduling a workflow.

task while it is executed on one instance, which is the total the cost for receiving data from the predecessor tasks.

4.3 Discrete Event-driven Simulation Process

For fitness evaluation, we develop a new discrete event-driven simulation process with the three rules, which maintains an event queue $\boldsymbol{\mathcal{E}}$ sorted with trigger time. Each event $\mathcal{E}_e$ is associated with the event name $\mathcal{E}_e^N$, trigger time $\mathcal{E}_e^T$, workflow involved $\mathcal{E}_e^W$, task involved $\mathcal{E}_e^A$, category involved $\mathcal{E}_e^\phi$, provider involved $\mathcal{E}_e^P$, and service instance involved $\mathcal{E}_e^S$. Fig. 5 shows the flowchart of dynamic event triggering when scheduling a workflow. According to the event name, we introduce (1) *when* each event is triggered and (2) *what* will happen when the event is triggered:

- `WorkflowSubmitted`. *When*: the arrival time of a workflow. *What*: set the tasks with no immediate predecessors to ready state. If there are multiple workflow submissions or multiple tasks that need to be set to ready state, the *task selection rule* is adopted to calculate the priority of each task, and the tasks are added to the `TaskReady` event queue in ascending order of this priority.
- `TaskReady`. *When*: after a workflow has been submitted or all of a task's immediate predecessors have been assigned and started to execute. *What*: select an instance to execute the task. We first use *cloud selection rule* to calculate the priority of each cloud. Then, in the cloud with the highest priority, one instance is selected to execute the task according to the *instance selection rule* introduced above. Once the instance that will execute the task has been determined, we can determine the start execution time of the task from the terminals associated with that instance.
- `StartBootingInstance`. *When*: before the instance executes its first task. *What*: take time to launch the instance and deploy the execution environments of workflows.
- `CompleteBootingInstance`. *When*: after the instance has finished booting. *What*: the instance enters an active state and can execute tasks. The different categories of services have different booting times.
- `StartDataTransfer`. *When*: after the predecessors of the task to be executed have finished execution. *What*: start to receive the output data from predecessors.
- `CompleteDataTransfer`. *When*: after receiving the output data of the predecessors. *What*: trigger the `StartExecuteTask` event if the output data of all predecessors are received.
- `StartExecuteTask`. *When*: after the ready task has received data from all predecessors and the assigned instance is active. *What*: start to execute the task, and set its successor to the ready state if all predecessors of this successor have started to execute. Generally, a task enters the ready state after its predecessor tasks are completed [2], [27], as shown by the dashed arrow ❶ in Fig. 5. However, in this case, if the instance executing the task needs to be created and initialised, it cannot receive data or start the task until the initialisation is complete. This can cause instances executing predecessor tasks to be idle unnecessarily. If we set a task to the ready state after starting to execute its predecessor tasks (the task's start time is after the completion of its predecessor tasks), as shown by the solid arrow ❷, we can initialise the instance earlier to reduce idle time.
- `CompleteExecuteTask`. *When*: after the execution of task is finished. *What*: trigger the `CompleteExecuteWorkflow` event if this task is the last finished task in a workflow.
- `CompleteExecuteWorkflow`. *When*: after all the tasks in a workflow have been finished. *What*: Record that the workflow has been scheduled.

The state of the simulated system, such as the system time, is updated as each event is triggered, and an event can generate and/or trigger other events. Overall, the simulation process is scheduled sequentially according to the event queue. The scheduling heuristic makes a decision on each decision point to make the scheduling process continue. The detailed pseudo-codes of the MTGP and discrete event-driven simulation process are presented in Section A of the supplementary file.

4.4 Complexity Analysis

Suppose that g is the number of evolutionary generations, p is the population size, n is the number of tasks in a decision point, c is the number of clouds, and s is the number of active service instances in a decision point. R_1, R_2, and R_3 are the number of nodes of the corresponding rule. The number of decisions at each of these three decision points does not

Table 3
Virtual Machine Category's Types and Configurations

	Type	vCPU	ECU	Total MIPS[a]	Memory (GB)	Cost($/hour)
AWS	c3.large	2	7	7000	3.75	0.128
	c3.xlarge	4	14	14000	7.5	0.255
	c3.2xlarge	8	28	28000	15	0.511
	c3.4xlarge	16	55	55000	30	1.021
	c3.8xlarge	32	108	108000	60	2.043
Azure	D2 v3	2	5	5000	8	0.096
	D4 v3	4	10	10000	16	0.192
	D8 v3	8	20	20000	32	0.384
	D16 v3	16	40	40000	64	0.768
	D32 v3	32	80	80000	128	1.536
Google	n1-standard-2	2	5.5	5500	7.5	0.095
	n1-standard-4	4	11	11000	15	0.19
	n1-standard-8	8	22	22000	30	0.38
	n1-standard-16	16	44	44000	60	0.76
	n1-standard-32	32	88	88000	120	1.52

[a] One ECU provides CPU capacity equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor, approximately 1000 MIPS.

Table 4
Function Category's Types and Configurations

	Type	Cost ($/ms)	Memory (MB)	vCPU	ECU[b]	$/10^6 *Reqs*	GHz	Total MIPS
AWS	1	2.9E-09	128	*0.072*[a]	*0.024*	0.2	\	*24.119*
	2	1.15E-08	512	*0.289*	*0.096*	0.2	\	*96.476*
	3	2.29E-08	1024	*0.579*	*0.193*	0.2	\	*192.953*
	4	4.58E-08	2048	*1.158*	*0.386*	0.2	\	*385.905*
	5	1.15E-07	5120	*2.894*	*0.965*	0.2	\	*964.764*
Google	1	2.31E-07	128	0.083	*0.028*	0.4	*0.035*[c]	*27.667*
	2	9.25E-07	512	0.333	*0.111*	0.4	*0.139*	*111.000*
	3	1.65E-06	1024	0.583	*0.194*	0.4	*0.243*	*194.333*
	4	2.90E-06	2048	1	*0.333*	0.4	*0.417*	*333.333*
	5	5.80E-06	4096	2	*0.667*	0.4	*0.833*	*666.667*

[a] At 1,769 MB, a function has the equivalent of one vCPU. https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html
[b] We assume a vCPU is roughly equivalent to 3.0 ECUs [52].
[c] 1 vCPU is equal to 2.4GHz. CPU. https://cloud.google.com/functions/pricing

Table 5
Public cloud cluster traces [57]

Cloud Vendor	Dataset	Released Year	Period	Content
Google[a]	ClusterData2011	2011	1 month	A single 12.5 k-machine Borg cell
	ClusterData2019	2019	1 month	Eight Borg cells
Azure[b]	PublicDatasetV1	2017		2 M VMs and 1.2 B utilisation readings
	PublicDatasetV2	2019		2.6 M VMs and 1.9 B utilisation readings
	FunctionsDataset2019	2019	2 weeks	A subset of applications running on Azure Functions
Alibaba[c]	Cluster-trace-v2017	2017	12 hours	About 1300 machines
	Cluster-trace-v2018	2018	8 days	About 4000 machines and 1.43×10^7 tasks, **the DAG information of production batch workloads**

[a] https://github.com/google/cluster-data
[b] https://github.com/Azure/AzurePublicDataset
[c] https://github.com/alibaba/clusterdata

exceed the total number of the tasks of all the submitted workflows in off-line training phase, denoted as N.

The complexity during the GP training phase is demonstrated by the fitness evaluation. The complexity in decision points is $O(n \cdot R_1) + O(c \cdot R_2) + O(s \cdot R_3) = O(n \cdot R_1 + c \cdot R_2 + s \cdot R_3)$. The complexity of fitness evaluation is $O(N \cdot n \cdot R_1 + N \cdot c \cdot R_2 + N \cdot s \cdot R_3)$. Therefore, for the offline training phase, the overall complexity of MTGP is $O\big(g \cdot p \cdot (O(N \cdot n \cdot R_1 + N \cdot c \cdot R_2 + N \cdot s \cdot R_3) + O(R_1 + R_2 + R_3))\big) = O(g \cdot p \cdot N \cdot n \cdot R_1 + g \cdot p \cdot N \cdot c \cdot R_2 + g \cdot p \cdot N \cdot s \cdot R_3)$. The complexity of online test phase is the same as the complexity in decision points, that is $O(n \cdot R_1 + c \cdot R_2 + s \cdot R_3)$.

5 Performance Evaluation and Results

5.1 Simulation Environment Setup

5.1.1 *Resource Environment*

To test on representative simulations close to real-world multi-cloud environments, we select three representative cloud providers and service categories respectively. The three providers are Amazon, Google and Microsoft. The three resource categories are VMs (provided by all three providers[2]), functions (provided by Amazon Lambda[3], and Google Cloud Functions[4]), and office365 (provided by Microsoft). The configurations of the virtual machine and functions, based on [22], [48], [52] and the corresponding official websites mentioned above, are shown in Table 3 and Table 4, respectively. Although there have been studies focusing on serverless workflow modelling for real-world platforms [15], [53], we are unable to follow those models due to problem characteristics and platform limitations. Therefore, we adopt a simplified model for AWS Lambda functions' computation capabilities, considering them proportional to allocated memory, based on AWS's resource allocation guidelines. This approach aligns with precedents in cloud workflow scheduling research [24], [54], [55] and facilitates our use of a simulation platform for task execution time estimation. All the three providers we considered support per-second billing, so we set the billing interval for VM category to 1 second[2]. The billing interval for the function category provided by Amazon is 1 ms[3], and the billing interval for the function category provided by Google is 100 ms[4]. The category office365 is not listed in the tables, since it is provided by only one provider. We assume that the execution cost of this category of task is fixed, and only consider the impact of task execution time on scheduling. Additionally, new instances can be started at any time for incoming tasks, regardless of office365 instance booting time.

We assume that the customer can lease any resource without any limitations [7], [22], and all required softwares for executing workflows can be installed on each leased resource category. It is worth mentioning that the function instances have a maximum of 900 seconds per execution[3]. If the execution time of a task on a function instance exceeds 900 seconds, the instance is unavailable. We set the boot time of a VM instance to 55.9 seconds [56], and the boot time of a function instance to 6.0 seconds. We set the average intra-cloud bandwidth to 0.1 GB/s, the average inter-cloud bandwidth to 0.05 GB/s and the unit price of data transmission to 0.02 $/GB.

5.1.2 *Workflow Applications*

We consider the real cluster data released by cloud computing vendors. At present, there are some large-scale enterprises that have disclosed some relevant data of cloud computing, such as Google, Azure, Alibaba, etc. The concise information of the data set is shown in the following Table 5. The cluster data released by Alibaba in 2018 [23] contains the

2. https://aws.amazon.com/ec2/pricing/on-demand/, https://cloud.google.com/compute/vm-instance-pricing, https://azure.microsoft.com/en-gb/pricing/, https://instances.vantage.sh/.
3. https://aws.amazon.com/cn/lambda/
4. https://cloud.google.com/functions/pricing

Table 6
The parameter settings of MTGP [30], [51].

Parameter	Value
Population size	200
Number of generations	50
Method for initialising population	Ramped-half-and-half
Initial minimum/maximum depth	2 / 7
Elitism	5
Maximal depth	8
Crossover rate	0.8
Mutation rate	0.15
Reproduction rate	0.05
Parent selection	Tournament selection with size 7
Terminal/non-terminal selection rate	10% / 90%

DAG information of workflow that obtains the interdependence between tasks, which is very consistent with the purpose of this study. Therefore, we select Alibaba cluster-trace-v2018 in this study. We extract 5200 DAG structures from the trace, where each DAG has at least 10 nodes. We also adopt five realistic scientific workflows (including Montage, CyberShake, Epigenomics, LIGO, and Sipht) commonly used in research to verify the performance of our algorithm. These data can be obtained from the workflow repository available at Pegasus website (https://download.pegasus.isi.edu/misc/SyntheticWorkflows.tar.gz).We choose workflows with 50, 100, 200 and 300 tasks, so we have $5\times4\times20 = 400$ workflows (5 workflow types, each with 4 different task numbers and 20 different data for workflows with different task numbers).

Scenario Construction: Although these public data provide the DAG structure and the execution time of each task, we need to construct data applicable to our problems based on the known information, such as the weight of the tasks and the number of invocations of the tasks. All the reconstructed data are available at https://github.com/zaixing-sun/MTGP_DFWS-MCs. We classify the tasks of each workflow into three categories 'VM', 'function', and 'office365' in a ratio of 60%, 35% and 5%. The input and output data of each task is assigned by a uniform discrete distribution between 500 and 5000 MB. Each DAG's deadline factor is randomly selected from the set $\{0.8, 1.0, 1.5, 1.8\}$.

Workflow arrival is modeled using the Poisson distribution with an arrive rate λ where the interarrival time is exponentially distributed with $1/\lambda$ [58], where $\lambda \in \{0.1, 0.2, 0.5, 0.8, 1.0\}$. For Cluster-trace-v2018, we randomly divide these 5200 DAGs into 8 datasets: 300-0, 300-1, 500-0, 500-1, 800-0, 800-1, 1000-0, and 1000-1. For scientific workflows, we divide these 400 DAGs into 2 datasets: 200-0 and 200-1. The datasets ending with '-0' are used for training, and the datasets ending with '-1'are used for test. We randomly select 10% the workflows from these 10 datasets to simulate dynamic scenarios.

The *name of each scenario* is set to <wfNum, λ>, where wfNum is the total number of workflows submitted and λ is the arrival rate. Therefore, there are $5\times4 = 20$ scenarios for training and test.

5.2 Baseline Algorithms and Evaluation Metrics

To evaluate the performance of the proposed MTGP algorithm, we evaluated it along with DWS [7], REMSM [21] and RMWS [27], which are three heuristic algorithms for solving the cost-minimising dynamic (real-time) workflow scheduling problem in the cloud. REMSM is indeed the only existing method for solving the cost-minimising dynamic (real-time) workflow scheduling problem in multi-clouds. Although DWS and RMWS are developed for single cloud rather than multi-cloud, we compare with them for the sake of more comprehensive comparison, as they are solving the most similar problem, and they are popular algorithms over the past five years. Moreover, they include task and instance selection decisions, which makes them suitable for making decisions in multi-clouds similar to REMSM. To make the comparison as fair as possible, we tried our best to adapt DWS and RMWS from single cloud to multi-cloud. **DWS** handles the dynamics of multiple deadline constrained workflows arriving randomly and schedules these workflows with reduced cost. In DWS, the task selection rule is based on the earliest sub-deadline first strategy and the instance selection rule is based on a cost-time trade-off factor. **REMSM** has better performance in the terms of the monetary cost for dynamic workflow in multi-cloud systems. In REMSM, the task selection rule is also based on the earliest sub-deadline first strategy. For instance selection rule, REMSM designs a balancing factor similar to that of DWS to select instance, but the difference is that REMSM gives priority to the instances that meet the sub-deadline. **RMWS** is a real-time multiple-workflow scheduling scheme to dynamically schedule workflows with minimum cost under different deadline constraints. RMWS designs a probability-based calculation method for sub-deadline of tasks, and feeds back the information from scheduled tasks to unscheduled tasks to update the sub-deadline. The instance selection rule is based on task's sub-deadline and the increased cost of the VM. This paper and all baseline algorithms have different calculation methods for sub-deadline division.

In addition, based on our proposed algorithm, we also compare the 2-decision MTGP with the 3-Decision MTGP. We name the proposed 3-Decision MTGP as **3D** and the 2-decision MTGP as **2D**. We use all terminals (used for constructing the cloud selection rule and the instance selection rule of 3D) to construct the instance selection rule of 2D. 2D has no cloud selection rule and the instance selection rule in 2D also follows 3D. Since the algorithms in [38] and [39] are based on the traditional GP framework, they need to consider the extra features of our problem by extending the terminal set so that they become 2D in our comparison algorithm. In other words, the comparison algorithm 2D is essentially the application of the algorithms of [38] and [39] to our problem.

The core scheduling goal of multiple-workflow scheduling in the cloud is to minimise the total monetary cost for executing workflows under deadline constraints. In addition, to better evaluate the scheduling performance of the algorithms, other indicators are considered as follows [27]:

1) Success rate: The success rate is the proportion of workflows that meet their scheduling deadline.

$$SR = \frac{||g \in \mathcal{G}|\max_{t\in T(g)} x(t) + \tau^e(t) \le \rho(t)||}{|\mathcal{G}|} \tag{17}$$

2) Deadline deviation: The deadline deviation is defined as the average duration percentage of exceeding work-

Table 7
The mean (standard deviation) Total Monetary Cost on the test data of 30 independent runs of MTGP and baseline methods for different scenarios.

<wfNum, λ>[a]	DWS	REMSM	RMWS	2D	3D
<20, 0.2>	762.85	1013.14(+)	861.02(0.00)(+)(-)	374.74(49.70)(-)(-)(-)	368.23(69.69)(-)(-)(-)(≈)[b]
<20, 0.5>	773.46	1017.62(+)	852.08(0.00)(+)(-)	383.00(65.33)(-)(-)(-)	371.46(59.07)(-)(-)(-)(≈)
<20, 0.8>	774.03	1018.08(+)	861.64(0.00)(+)(-)	399.83(92.15)(-)(-)(-)	348.68(39.54)(-)(-)(-)(-)
<20, 1.0>	770.33	1018.94(+)	861.88(0.00)(+)(-)	374.46(58.73)(-)(-)(-)	391.11(135.78)(-)(-)(-)(≈)
<30, 0.2>	779.40	448.25(-)	427.29(0.00)(-)(-)	431.71(3.43)(-)(-)(+)	420.02(19.00)(-)(-)(-)(-)
<30, 0.5>	779.71	446.67(-)	422.69(0.00)(-)(-)	430.33(3.16)(-)(-)(+)	415.00(3.09)(-)(-)(-)(-)
<30, 0.8>	779.32	448.13(-)	422.84(0.00)(-)(-)	431.43(6.95)(-)(-)(+)	419.10(13.76)(-)(-)(-)(-)
<30, 1.0>	779.56	448.54(-)	422.55(0.00)(-)(-)	430.10(3.09)(-)(-)(+)	416.03(9.24)(-)(-)(-)(-)
<50, 0.2>	1459.43	948.04(-)	1005.33(0.00)(-)(+)	967.16(79.57)(-)(≈)(-)	897.49(5.09)(-)(-)(-)(-)
<50, 0.5>	1467.81	947.45(-)	944.49(0.00)(-)(-)	947.09(14.41)(-)(-)(≈)	900.96(11.71)(-)(-)(-)(-)
<50, 0.8>	1465.29	946.74(-)	941.86(0.00)(-)(-)	949.32(24.63)(-)(+)(+)	899.55(9.48)(-)(-)(-)(-)
<50, 1.0>	1466.21	949.75(-)	938.91(0.00)(-)(-)	957.72(62.26)(-)(+)(+)	897.22(3.97)(-)(-)(-)(-)
<80, 0.2>	1857.49	1266.24(-)	1336.58(0.00)(-)(+)	1366.17(16.51)(-)(+)(+)	1243.68(41.51)(-)(-)(-)(-)
<80, 0.5>	1864.49	1275.71(-)	1288.46(0.00)(-)(+)	1366.55(13.97)(-)(+)(+)	1243.99(40.79)(-)(-)(-)(-)
<80, 0.8>	1872.60	1282.47(-)	1273.36(0.00)(-)(-)	1368.31(19.06)(-)(+)(+)	1238.22(25.44)(-)(-)(-)(-)
<80, 1.0>	1867.92	1274.92(-)	1288.32(0.00)(-)(+)	1363.08(6.75)(-)(+)(+)	1235.46(8.80)(-)(-)(-)(-)
<100, 0.2>	3435.03	2427.42(-)	2508.94(0.00)(-)(+)	2512.69(18.04)(-)(+)(≈)	2302.71(7.92)(-)(-)(-)(-)
<100, 0.5>	3445.77	2426.11(-)	2409.24(0.00)(-)(-)	2506.44(7.05)(-)(+)(+)	2299.77(4.72)(-)(-)(-)(-)
<100, 0.8>	3448.49	2428.72(-)	2362.35(0.00)(-)(-)	2509.21(11.07)(-)(+)(+)	2298.38(3.33)(-)(-)(-)(-)
<100, 1.0>	3453.03	2426.89(-)	2341.21(0.00)(-)(-)	2507.28(7.64)(-)(+)(+)	2309.49(27.07)(-)(-)(-)(-)
Win/Draw/Lose	0/0/20	0/0/20	0/0/20	0/3/17	N/A
Average Rank	4.6	3.49	2.74	3.04	**1.13**

[a] wfNum is the total number of workflows submitted and λ is the arrival rate.
[b] (-)(-)(-)(≈) indicates that 3D is significantly better than DWS, REMSM, and RMWS, while it has no significant difference from 2D.

flow deadlines in workflow set G.

$$DD = \sum_{g\in\mathcal{G}} \frac{\max\limits_{t\in T(g)} \{x(t)+\tau^e(t)\}+a(g)-\rho(g)}{\rho(g)-a(g)} \Bigg/ |\mathcal{G}| \quad (18)$$

We implement the simulation and MTGP in Python 3.10 and deploy it on a computer with a 3.80 GHz Intel Core i7-10700K processor and 15.3 GB of memory. The parameters of the MTGP are shown in Table 6.

For all the compared algorithms, 30 independent runs are done for each scenario and the evolved scheduling heuristics are tested on 50 unseen instances. The average objective value across the 50 test instances is reported as the test performance of the rule, which can be a good approximation of the true performance of the rule.

Friedman's test with a significance level of 0.05 is applied to rank the algorithms based on their performance. If Friedman's test gives significance results, we further conduct the Wilcoxon rank-sum test with Bonferroni correction between the proposed algorithm and other algorithms with a significance level of 0.05 for the Nemenyi post-hoc pairwise comparisons [35], [59], [60]. In the following results, "-", "+", and "≈" indicate that the corresponding result is significantly better than, worse than, or similar to its counterpart. An algorithm will be compared with the algorithm(s) before it one by one. For the values of *Total Monetary Cost* and *Deadline Deviation*, the smaller the better performance. For the value of *Success Rate*, the bigger the better performance. "Win, Draw, Lose" means the number of scenarios that a compared algorithm is statistically better, similar, or worse than 3D. "Average Rank" shows the average ranking of the algorithm on all the examined scenarios.

5.3 Results

The runtime of the proposed method is acceptable in dynamic scheduling scenarios. Our algorithm mainly involves three decision points, i.e. task/cloud/instance selection, and the average decision-making time of each decision point in the online test phase is less than 0.02 seconds. The runtime data and analysis of scheduling algorithms are tabulated in Section B of the supplementary file.

Table 7 shows the mean and standard deviation of the monetary cost on the test data of 30 independent runs of MTGP and the baseline methods for different scenarios. Except that there is no significant difference between 3D and 2D in scenarios $<20, 0.2>$, $<20, 0.5>$ and $<20, 1.0>$ (which we analyse in the next section), the proposed 3D method performs significantly better than all the baseline methods in all other scenarios. For scientific workflows (wfNum=20), DWS is the best baseline algorithms, but the 2D and 3D algorithms still outperform it by more than 45%. The results also show that 3D performs the best with the smallest rank based on the average ranking according to the Friedman test, and is significantly better than the algorithms in most of the scenarios. RMWS is the second-best among all other algorithms (Average Rank=2.74). This demonstrates that the scheduling heuristics learned by MTGP performs significantly better than other human designed heuristics on the scientific workflows. For Cluster-trace-v2018, 3D significantly outperforms the other algorithms, followed by RMWS, while the performance of DWS becomes worse. For the same type of scientific workflows, their structures are often similar. The workflows in Cluster-trace-v2018 do not have a fixed DAG structure. This suggests that 3D is more adaptable, while DWS is highly dataset dependent. As the number of workflows increases, the performance of 2D becomes progressively weaker than that of REMSM. Since the other algorithms have 2 rules, i.e. only task and instance selection rules, except for 3D which has 3 rules, we can also conclude that learning three rules simultaneously is more effective than (learning) two rules only.

DWS and REMSM do not have standard deviation values because they are deterministic heuristic algorithms. The standard deviation of RMWS is 0.00 (round it to 0.01, which

Table 8
The mean (standard deviation) Deadline Deviation and Success Rate on the test data of 30 independent runs of MTGP and baseline methods for different scenarios.

<wfNum, λ>	Deadline Deviation					Success Rate				
	DWS	REMSM	RMWS	2D	3D	DWS	REMSM	RMWS	2D	3D
<20, 0.2>	-0.35	0.01(+)	0.01(+)(+)	-0.07(0.06)(+)(-)(-)	-0.04(0.02)(+)(-)(-)(≈)	0.89	0.42	0.39	**0.97**	0.95
<20, 0.5>	-0.35	0.01(+)	0.02(+)(+)	-0.08(0.06)(+)(-)(-)	-0.04(0.02)(+)(-)(-)(+)	0.89	0.41	0.40	**0.97**	0.94
<20, 0.8>	-0.35	0.01(+)	0.02(+)(+)	-0.06(0.05)(+)(-)(-)	-0.04(0.01)(+)(-)(-)(+)	0.89	0.39	0.38	**0.97**	0.95
<20, 1.0>	-0.35	0.01(+)	0.02(+)(+)	-0.07(0.06)(+)(-)(-)	-0.04(0.01)(+)(-)(-)(+)	0.89	0.43	0.37	**0.97**	0.94
<30, 0.2>	-0.56	-0.14(+)	-0.12(+)(+)	-0.14(0.03)(+)(≈)(-)	-0.18(0.07)(+)(-)(-)(-)	0.98	0.97	0.94	**0.99**	**0.99**
<30, 0.5>	-0.55	-0.13(+)	-0.11(+)(+)	-0.13(0.03)(+)(≈)(-)	-0.19(0.06)(+)(-)(-)(-)	0.98	0.97	0.95	**0.99**	**0.99**
<30, 0.8>	-0.55	-0.13(+)	-0.12(+)(+)	-0.14(0.03)(+)(≈)(-)	-0.19(0.05)(+)(-)(-)(-)	0.98	0.97	0.95	**0.99**	**0.99**
<30, 1.0>	-0.55	-0.13(+)	-0.12(+)(+)	-0.13(0.03)(+)(≈)(≈)	-0.19(0.05)(+)(-)(-)(-)	0.98	0.97	0.95	**0.99**	**0.99**
<50, 0.2>	-0.54	-0.12(+)	-0.11(+)(+)	-0.14(0.05)(+)(≈)(-)	-0.26(0.09)(+)(-)(-)(-)	0.97	0.96	0.96	**1.00**	0.99
<50, 0.5>	-0.54	-0.13(+)	-0.10(+)(+)	-0.14(0.04)(+)(≈)(-)	-0.25(0.06)(+)(-)(-)(-)	0.97	0.96	0.96	**1.00**	**1.00**
<50, 0.8>	-0.54	-0.12(+)	-0.11(+)(+)	-0.15(0.06)(+)(-)(-)	-0.25(0.06)(+)(-)(-)(-)	0.97	0.95	0.95	**1.00**	0.99
<50, 1.0>	-0.55	-0.13(+)	-0.11(+)(+)	-0.16(0.07)(+)(-)(-)	-0.25(0.07)(+)(-)(-)(-)	0.97	0.96	0.95	**1.00**	0.99
<80, 0.2>	-0.53	-0.10(+)	-0.10(+)(+)	-0.16(0.07)(+)(-)(-)	-0.28(0.08)(+)(-)(-)(-)	0.97	0.95	0.97	**0.99**	**0.99**
<80, 0.5>	-0.53	-0.10(+)	-0.09(+)(+)	-0.18(0.07)(+)(-)(-)	-0.31(0.08)(+)(-)(-)(-)	0.97	0.94	0.95	**0.99**	**0.99**
<80, 0.8>	-0.53	-0.10(+)	-0.09(+)(+)	-0.22(0.11)(+)(-)(-)	-0.29(0.06)(+)(-)(-)(-)	0.97	0.94	0.95	**0.99**	**0.99**
<80, 1.0>	-0.53	-0.10(+)	-0.09(+)(+)	-0.18(0.07)(+)(-)(-)	-0.29(0.07)(+)(-)(-)(-)	0.97	0.94	0.94	**0.99**	**0.99**
<100, 0.2>	-0.53	-0.10(+)	-0.09(+)(+)	-0.15(0.04)(+)(-)(-)	-0.23(0.07)(+)(-)(-)(-)	0.96	0.96	0.97	**0.99**	**0.99**
<100, 0.5>	-0.53	-0.10(+)	-0.09(+)(+)	-0.15(0.05)(+)(-)(-)	-0.23(0.07)(+)(-)(-)(-)	0.96	0.96	0.96	**0.99**	**0.99**
<100, 0.8>	-0.53	-0.10(+)	-0.09(+)(+)	-0.14(0.04)(+)(-)(-)	-0.26(0.07)(+)(-)(-)(-)	0.96	0.96	0.95	**0.99**	**0.99**
<100, 1.0>	-0.53	-0.10(+)	-0.09(+)(+)	-0.16(0.08)(+)(-)(-)	-0.25(0.06)(+)(-)(-)(-)	0.96	0.96	0.96	**0.99**	**0.99**
Win/Draw/Lose	20/0/0	0/0/20	0/0/20	3/1/16	N/A	0/0/20	0/0/20	0/0/20	7/13/0	N/A
Average Rank	**1.0**	3.69	4.82	3.17	2.33	3.2	4.22	4.58	**1.02**	1.98

is actually close to 0), which mainly because the algorithm only introduces a probability-based Boolean variable indicating whether to consider the data transfer time, when calculating the probabilistic upward rank. The results show that the setting of this part has little effect on the performance of the algorithm.

Table 8 shows the mean Deadline Deviation and Success Rate on test data of 30 independent runs of MTGP and baseline methods for different scenarios. The results show that DWS achieves the smallest Deadline Deviation compared to other algorithms, indicating that the cost-time trade-off factor designed by this algorithm is more focused on time, and consequently obtains a higher success rate. The main reason is that DWS's task selection rule is based on the earliest deadline first strategy, and always prioritises tasks having the nearest deadline, but tasks at the same level on the same workflow have the same sub-deadlines. However, for DWS, tasks at the same level on the same workflow have the same sub-deadline. Nevertheless, DWS cannot select more suitable instances to execute tasks on the Cluster-trace-2018 datasets (as it has the highest cost). The main reason for this is that DWS relies absolutely on the cost-time trade-off factor to select instances and ignores whether the selected instance meets the sub-deadline or not. 2D and 3D have their own advantages for different types of datasets, but both are significantly better than REMSM and RMWS. The results in Table 8 also show that 2D and 3D always achieve higher or similar Success Rate, especially reaching almost 100% on the Cluster-trace-2018 datasets. This result confirms the performance of 2D and 3D on Deadline Deviation from another aspect. The performance of 3D is always better on Total Monetary Cost and Success Rate metrics, and only slightly worse than DWS on Deadline Deviation, mainly due to the trade-off between cost and time. 3D's instance selection rule is to prefer those instances with sub-deadlines that can be met, whereas our goal is to minimise the total cost, so the rules learned by 3D tend to select instances that have less cost and meet the sub-deadlines.

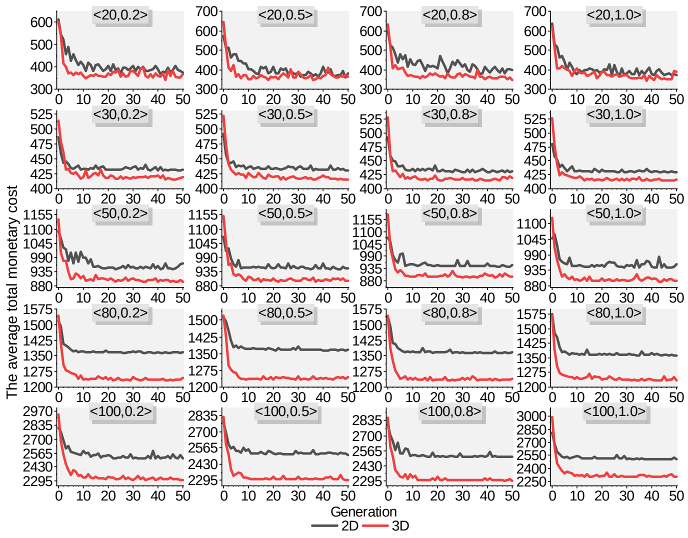


Fig. 6. The curves of the average total monetary cost of 2D and 3D over 30 independent runs on the test datasets of all the scenarios.

As the arrival rate increases, the performance of all the algorithms change very slightly on the three metrics considered, indicating that they can take advantage of the elastic properties of the cloud to adapt to changes in the environment.

In summary, the proposed 3D algorithm can achieve better performance in all scenarios based on scientific workflows and Cluster-trace-2018 datasets, which further suggests that the learned heuristic rules are more effective than manually designed heuristic rules. It also achieves better performance than 2D in the Cluster-trace-2018 dataset.

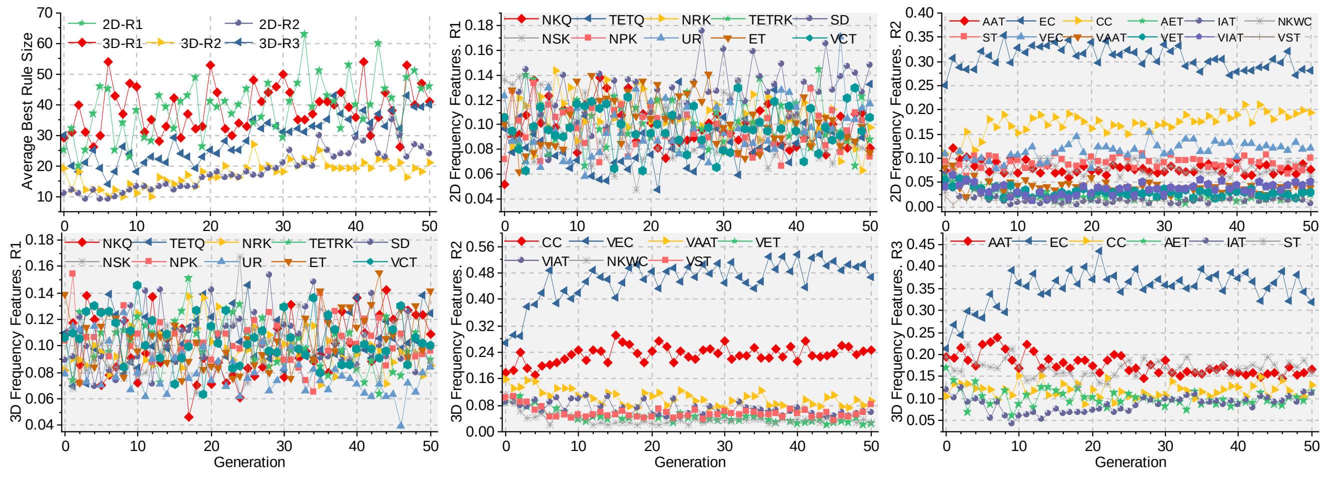
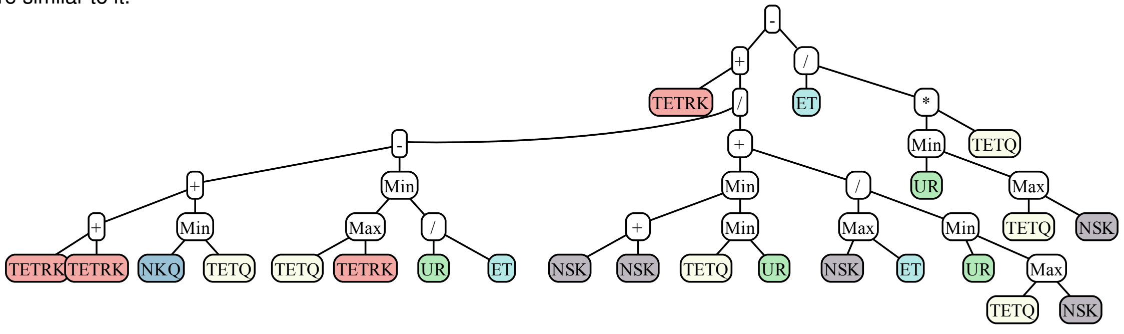
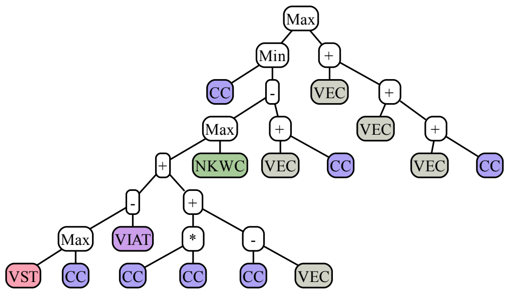


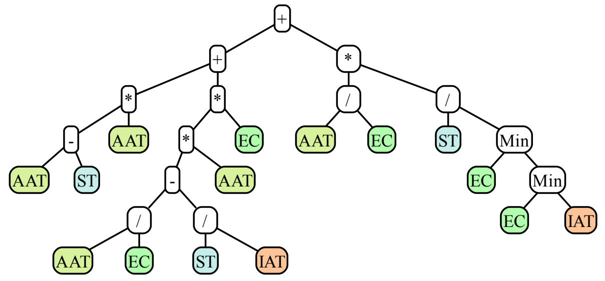
Fig. 7. The curves of the average best different rule sizes and feature frequency of 2D and 3D over 30 independent runs in the scenario $<50, 0.2>$. 2D-R1 is **task selection rule** of 2D; 2D-R2 is **instance selection rule** of 2D; 3D-R1 is *task selection rule* of 3D; 3D-R2 is *cloud selection rule* of 3D; 3D-R3 is *instance selection rule* of 3D. '*2D Frequency Features. R1*' is the frequency features of the task selection rule of the 2D algorithm, and the others are similar to it.



(a) An example of task selection rule evolved by 3-Decision MTGP.



(b) An example of cloud selection rule evolved by 3-Decision MTGP.



(c) An example of instance selection rule evolved by 3-Decision MTGP.

Fig. 8. Examples of learned rules evolved by 3-Decision MTGP.

5.4 3-Decision MTGP *Versus* 2-Decision MTGP

Fig. 6 shows the curves of the average cost of 2D and 3D over 30 independent runs on all scenarios test datasets. The results show that although 3D has a higher cost in most scenarios at generation 0, as the algorithms evolve, the cost of 3D quickly becomes much lower than that of 2D. Due to the dynamic scheduling, their curves fluctuate but generally show a downward convergence trend. For scientific workflows (wfNum=20), the gap between 2D and 3D is smaller and their fluctuations are larger. Although there is no significant difference between them in the three scenarios, the curves of 3D are generally superior to those of 2D. We can also see that as the number of workflows increases, the gap between 2D and 3D becomes more significant and their fluctuations become smaller.

To investigate why 3D is better than 2D in most scenarios, we choose scenario $<\text{wfNum}, \lambda> = <50, 0.2>$ to carry out further analysis since other scenarios have similar results. Fig. 7 shows the curves of the average best different rule sizes and feature frequency of 2D and 3D over 30 independent runs in the scenario $<50, 0.2>$. Although the sizes of the task selection rules (R1) for 2D and 3D are always the largest, their feature frequencies seem to always fluctuate and there is no obvious stratification to specifically select certain terminals. This suggests that these features play a similar role in the construction of task selection rules. The feature frequency plots of the instance selection rules and the cloud selection rule show an obvious stratification as the algorithm evolves, indicating that these terminals with high feature frequencies help the algorithm to make efficient decisions. Although 3D's cloud selection rule almost always has the smallest size, its two terminators, VEC and CC,

$$R1 = \mathrm{TETRK} + \frac{2\mathrm{TETRK} + \min\{\mathrm{NKQ}, \mathrm{TETQ}\} - \min\left\{\max\{\mathrm{TETQ}, \mathrm{TETRK}\}, \frac{\mathrm{UR}}{\mathrm{ET}}\right\}}{\min\{2\mathrm{NSK}, \mathrm{TETQ}, \mathrm{UR}\} + \frac{\max\{\mathrm{NSK}, \mathrm{ET}\}}{\min\{\mathrm{UR}, \max\{\mathrm{TETQ}, \mathrm{NSK}\}\}}} - \frac{\mathrm{ET}}{\min\{\mathrm{UR}, \max\{\mathrm{TETQ}, \mathrm{NSK}\}\} \times \mathrm{TETQ}} \tag{19}$$

$$R2 = \max\left\{\min\left\{\mathrm{CC}, \max\left\{\max\{\mathrm{VST}, \mathrm{CC}\} - \mathrm{VIAT} + \mathrm{CC}^2 + \mathrm{CC} - \mathrm{VEC}, \mathrm{NKWC}\right\} - \mathrm{VEC} - \mathrm{CC}\right\}, 3\mathrm{VEC} + \mathrm{CC}\right\} \tag{20}$$

$$R3 = (\mathrm{AAT} - \mathrm{ST}) \times \mathrm{AAT} + (\frac{\mathrm{AAT}}{\mathrm{EC}} - \frac{\mathrm{ST}}{\mathrm{IAT}}) \times \mathrm{AAT} \times \mathrm{EC} + \frac{\mathrm{AAT}}{\mathrm{EC}} \times \frac{\mathrm{ST}}{\min\{\mathrm{EC}, \mathrm{IAT}\}} \tag{21}$$

accounted for a total of 46.6% + 24.6% = 71.2% in the last generation. This shows that 3D tends to select clouds with smaller VEC and CC in each cloud. The size of the instance selection rules for 3D is significantly larger than for 2D, especially in the last generation where there are on average 40-24=16 more nodes. For the characteristic frequencies, EC, AAT and ST in 3D account for 31.8%, 16.6% and 15.9% respectively, while EC, CC, VEC and ST in 2D account for 28.0%, 19.4%, 12.2% and 10% respectively. The results show that 3D not only focuses on EC, but also considers time-related terminals in instance selection to select instance that are suitable in terms of cost and time to execute tasks. In 2D, there is no cloud selection rule and it has to focus on cost (EC and CC).

In general, compared to 2D that selects a resource from all available resources with larger selection space, the advantage of 3D lies in its ability to first consider the characteristics and states of each cloud as a whole, and to select a cloud that has lower overall execution and transmission costs. Within this selected cloud with a smaller selection space, 3D learns and uses more valuable terminals (such as AAT and ST) so that it can select a resource that is more suitable for the task in terms of time and cost.

5.5 Insight of evolved scheduling heuristics

To gain further understanding of the behaviour of the scheduling heuristic evolved by the proposed method, an evolved scheduling heuristic is selected to be analysed. Fig. 8 shows three rules from the selected scheduling heuristic evolved by MTGP in scenario ($<\mathrm{wfNum}, \lambda>$ = $<50, 0.2>$). The selected scheduling heuristic has a promising test performance.

Fig. 8(a) shows that the task selection rule is a combination of six terminals (TETRK, NKQ, TETQ, UR, ET and NSK), where TETQ and NSK are the most frequently used terminals in this rule. The rule can be simplified as $R1$, as shown in Eq. (19). We can know that this rule prioritises workflow with a longer total execution time in ready queue and task with more successors. Fig. 8(b) shows that the cloud selection rule is a combination of five terminals (CC, VEC, VST, NKWC and VIAT), where CC and VEC are the most frequently used terminals in this rule. The rule can be simplified as $R2$, as shown in Eq. (20). We can know that this rule mainly relies on the interaction between communication cost and average execution cost to select cloud. Fig. 8(c) shows that the instance selection rule is a combination of four terminals (AAT, EC, ST and IAT), where AAT and EC are the most frequently used terminals in this rule. The rule can be simplified as $R3$, as shown in Eq. (21). We can know that this rule prioritises instances with smaller actual available time and execution cost to execute tasks. These three rules are consistent with the observations of section 5.4.

Results show that dynamic workflow scheduling not only needs easy-to-find factor design algorithms such as EC, CC and SD, but also needs to dissect the scheduling process and extract factors that can reflect the system state in detail, such as TETQ and AAT. In addition, scheduling heuristic rules learned through algorithm evolution are more effective than those manually designed rules.

6 Conclusions

This paper considers a more near-realistic dynamic flexible workflow scheduling in multi-clouds, where each cloud can provide different service categories and the workflow's flexibility lies in its tasks being associated with specific service categories. To the best of our knowledge, this is the first work on dynamic workflow scheduling in multi-clouds with multiple service categories. To solve the problem, we develop a dynamic scheduling simulator to simulates the workflow scheduling process in real-world scenarios, which employs three scheduling heuristics to make decisions at three decision points to keep scheduling going. The proposed 3-Decision MTGP method can evolve the three scheduling heuristics (i.e., task selection rule, cloud selection rule and instance selection rule) simultaneously to meet these decision points. 3-Decision MTGP is examined based on two real-world data traces. The simulation results and comparisons show that it can achieve up to 45% reduction in terms of total monetary cost compared to existing algorithms in scientific workflow datasets, and it performs significantly better than all baseline algorithms and its variant algorithm (2-Decision MTGP) in all scenarios based on Cluster-trace-2018 datasets. Further analyses show that the superiority of *3-Decision MTGP* is realised by selecting a cloud that has lower overall execution and transmission costs, and then learning more valuable terminals (such as AAT and ST) to select a more suitable resource for a task in terms of time and cost.

In the future, we will consider applying this MTGP method to solve multi-objective DFWS-MCs problems in terms of reliability, rental cost and energy consumption. For workflow scheduling in cloud environments, there may be a large number of features, whose importance/relevance varies from one to another. This work already shows the importance of some features in different rules. Therefore, we would like to investigate GPHH with feature selection to find more promising features to schedule tasks properly and further improve its performance. We also intend to analyse the gaps between the dynamic scheduling simulator and real-world scenarios.

References

[1] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-Time Tasks Oriented Energy-Aware Scheduling in Virtualized Clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 168–180, 2014.

[2] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-Aware Online Scheduling for Real-Time Workflows in Cloud Service Environment," *IEEE Trans. Serv. Comput.*, vol. 14, no. 4, pp. 1167–1178, 2021.

[3] T. Luxner, "Cloud computing trends and statistics: Flexera 2023 state of the cloud report," https://www.flexera.com/blog/cloud/cloud-computing-trends-flexera-2023-state-of-the-cloud-report/, accessed on 2023-08-04.

[4] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Cost-Effective Web Application Replication and Deployment in Multi-Cloud Environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1982–1995, 2022.

[5] Y. Xie, F. X. Gui, W. J. Wang, and C. F. Chien, "A Two-stage Multi-population Genetic Algorithm with Heuristics for Workflow Scheduling in Heterogeneous Distributed Computing Environments," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1446–1460, 2023.

[6] Q. Z. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, "A Cooperative Coevolution Hyper-Heuristic Framework for Workflow Scheduling Problem," *IEEE Trans. Serv. Comput.*, vol. 15, no. 1, pp. 150–163, 2022.

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

[8] S. A. Ahson and M. Ilyas, *Cloud computing and software services: Theory and techniques*, 1st ed. USA: CRC Press, Inc., 2010.

[9] S. Wang, X. Li, Q. Z. Sheng, and A. Beheshti, "Performance Analysis and Optimization on Scheduling Stochastic Cloud Service Requests: A Survey," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 3, pp. 3587–3602, 2022.

[10] S. Qin, D. Pi, Z. Shao, Y. Xu, and Y. Chen, "Reliability-Aware Multi-Objective Memetic Algorithm for Workflow Scheduling Problem in Multi-Cloud System," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1343–1361, 2023.

[11] Q. H. Zhu, H. Tang, J. J. Huang, and Y. Hou, "Task Scheduling for Multi-Cloud Computing Subject to Security and Reliability Constraints," *IEEE-CAA J. Automatica Sin.*, vol. 8, no. 4, pp. 848–865, 2021.

[12] S. Sharma and B. Sharma, "Optimal selection of application loading on cloud services," *Int. J. Prod. Res.*, vol. 54, no. 21, pp. 6512–6531, 2016.

[13] S. Sharif, P. Watson, J. Taheri, S. Nepal, and A. Y. Zomaya, "Privacy-Aware Scheduling SaaS in High Performance Computing Environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1176–1188, 2017.

[14] Z. Sun, H. Huang, Z. Li, C. Gu, R. Xie, and B. Qian, "Efficient, economical and energy-saving multi-workflow scheduling in hybrid cloud," *Expert Syst. Appl.*, p. 120401, 2023.

[15] C. Lin and H. Khazaei, "Modeling and Optimization of Performance and Cost of Serverless Applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 615–632, 2021.

[16] J. Kijak, P. Martyna, M. Pawlik, B. Balis, and M. Malawski, "Challenges for Scheduling Scientific Workflows on Cloud Functions," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 460–467.

[17] W. Hu, X. Li, and X. Li, "Hybrid Cloud Workflow Scheduling Method with Privacy Data," *IEEE Access*, vol. 8, pp. 211 540–211 552, 2020.

[18] C. Li, J. Tang, and Y. Luo, "Hybrid Cloud Adaptive Scheduling Strategy for Heterogeneous Workloads," *J. Comput.*, vol. 17, no. 3, pp. 419–446, 2019.

[19] Z. Li, J. Ge, H. Yang, L. Huang, H. Hu, H. Hu, and B. Luo, "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Futur. Gener. Comp. Syst.*, vol. 65, pp. 140–152, 2016.

[20] Z. Chen, K. Lin, B. Lin, X. Chen, X. Zheng, and C. Rong, "Adaptive Resource Allocation and Consolidation for Scientific Workflow Scheduling in Multi-Cloud Environments," *IEEE Access*, vol. 8, pp. 190 173–190 183, 2020.

[21] A. Taghinezhad-niar, J. Taheri, and S. Member, "Reliability, Rental-Cost and Energy-Aware Multi-Workflow Scheduling on Multi-Cloud Systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2681–2692, 2023.

[22] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, and H. Huang, "ET2FA: A Hybrid Heuristic Algorithm for Deadline-constrained Workflow Scheduling in Cloud," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 1807–1821, 2023.

[23] L. Ye, Y. Xia, L. Yang, and C. Yan, "SHWS: Stochastic Hybrid Workflows Dynamic Scheduling in Cloud Container Services," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 3, pp. 2620–2636, 2022.

[24] R. Xie, D. Gu, Q. Tang, T. Huang, and F. R. Yu, "Workflow Scheduling in Serverless Edge Computing for the Industrial Internet of Things: A Learning Approach," *IEEE Trans. Ind. Informatics*, vol. 19, no. 7, pp. 8242–8252, 2023.

[25] A. Mampage, S. Karunasekera, and R. Buyya, "A Holistic View on Resource Management in Serverless Computing Environments: Taxonomy and Future Directions," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–36, 2022.

[26] X. Tang, "Reliability-Aware Cost-Efficient Scientific Workflows Scheduling Strategy on Multi-Cloud Systems," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2909–2919, 2022.

[27] H. Xu, H. Gao, M. Bian, X. Ma, H. Xu, H. Gao, and M. Bian, "Real-Time Multiple-Workflow Scheduling in Cloud Environments," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 4, pp. 4002–4018, 2021.

[28] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An Efficient Feature Selection Algorithm for Evolving Job Shop Scheduling Rules with Genetic Programming," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 1, no. 5, pp. 339–353, 2017.

[29] 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, 2021.

[30] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Zhang, and M. Zhang, "Genetic Programming for Dynamic Workflow Scheduling in Fog Computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 2657–2671, 2023.

[31] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.

[32] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex Intell. Syst.*, vol. 3, no. 1, pp. 41–66, 2017.

[33] 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.

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

[35] 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. 10 515–10 528, 2022.

[36] 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, 2022.

[37] S. Wang, Y. Mei, and M. Zhang, "Explaining Genetic Programming-Evolved Routing Policies for Uncertain Capacitated Arc Routing Problems," *IEEE Trans. Evol. Comput.*, pp. 1–15, 2023.

[38] 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.*, vol. 12392 LNCS, 2020, pp. 76–90.

[39] 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.

[40] 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, 2002.

[41] H. Djigal, J. Feng, J. Lu, and J. Ge, "IPPTS: An Efficient Algorithm for Scientific Workflow Scheduling in Heterogeneous Computing Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1057–1071, 2021.

[42] I. Casas, J. Taheri, R. Ranjan, L. Wang, and A. Y. Zomaya, "A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems," *Futur. Gener. Comput. Syst.*, vol. 74, pp. 168–178, 2017.

[43] X. Xia, H. Qiu, X. Xu, and Y. Zhang, "Multi-objective workflow scheduling based on genetic algorithm in cloud environment," *Inf. Sci.*, vol. 606, pp. 38–59, 2022.

[44] P. Cong, G. Xu, J. Zhou, M. Chen, T. Wei, and M. Qiu, "Personality- and Value-Aware Scheduling of User Requests in Cloud for Profit Maximization," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1991–2004, 2022.

[45] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource Scheduling in Edge Computing: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[46] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling Internet of Things requests to minimize latency in hybrid Fog–Cloud computing," *Futur. Gener. Comput. Syst.*, vol. 111, pp. 539–551, 2020.

[47] R. Ranjan, I. S. Thakur, G. S. Aujla, N. Kumar, and A. Y. Zomaya, "Energy-Efficient Workflow Scheduling Using Container-Based Virtualization in Software-Defined Data Centers," *IEEE Trans. Ind. Informatics*, vol. 16, no. 12, pp. 7646–7657, 2020.

[48] M. Pawlik, P. Banach, and M. Malawski, "Adaptation of Workflow Application Scheduling Algorithm to Serverless Infrastructure," in *Proc. Euro-Par 2019 Parallel Process. Work.*, 2020, pp. 345–356.

[49] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J. Netw. Comput. Appl.*, vol. 143, pp. 1–33, 2019.

[50] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, *Genetic Programming for Production Scheduling*, 1st ed., ser. Machine Learning: Foundations, Methodologies, and Applications. Singapore: Springer Singapore, 2021.

[51] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Task Relatedness Based Multitask Genetic Programming for Dynamic Flexible Job Shop Scheduling," *IEEE Trans. Evol. Comput.*, pp. 1–15, 2022.

[52] M. Barika, S. Garg, A. Chan, and R. N. Calheiros, "Scheduling Algorithms for Efficient Execution of Stream Workflow Applications in Multicloud Environments," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 860–875, 2022.

[53] W. Wang, Q. Wu, Z. Zhang, J. Zeng, X. Zhang, and M. Zhou, "A probabilistic modeling and evolutionary optimization approach for serverless workflow configuration," *Softw. Pract. Exp.*, sep 2023.

[54] J. Zhang, X. Li, L. Chen, and R. Ruiz, "Scheduling Workflows With Limited Budget to Cloud Server and Serverless Resources," *IEEE Trans. Serv. Comput.*, pp. 1–14, 2023.

[55] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "Astrea: Auto-Serverless Analytics Towards Cost-Efficiency and QoS-Awareness," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3833–3849, 2022.

[56] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 423–430.

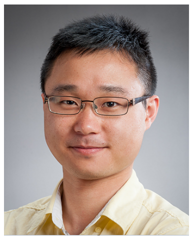
[57] J. Yu, M. Gao, Y. Li, Z. Zhang, W. H. Ip, and K. L. Yung, "Workflow performance prediction based on graph structure aware deep attention neural network," *J. Ind. Inf. Integr.*, vol. 27, p. 100337, 2022.

[58] S. Sahoo, B. Sahoo, and A. K. Turuk, "A Learning Automata-Based Scheduling for Deadline Sensitive Task in the Cloud," *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1662–1674, 2021.

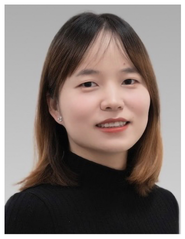
[59] Z. Zhang, Y. Fu, K. Gao, H. Zhang, and L. Wang, "A cooperative evolutionary algorithm with simulated annealing for integrated scheduling of distributed flexible job shops and distribution," *Swarm Evol. Comput.*, vol. 85, p. 101467, 2024.

[60] Z. J. Wang, Z. H. Zhan, W. J. Yu, Y. Lin, J. Zhang, T. L. Gu, and J. Zhang, "Dynamic Group Learning Distributed Particle Swarm Optimization for Large-Scale Optimization and Its Application in Cloud Workflow Scheduling," *IEEE Trans. Cybern.*, vol. 50, no. 6, pp. 2715–2729, 2020.

Zaixing Sun received the B.E. degree from Luoyang Normal University, Luoyang in 2016, and M.E. degree from Kunming University of Science and Technology, Kunming, China, in 2019. He is currently pursuing a PhD degree in computer science and technology with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing, intelligent optimisation and scheduling.



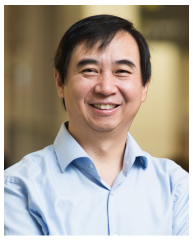
Yi Mei received the B.Sc. and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively. He is an Associate Professor at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation for combinatorial optimisation, genetic programming, and hyper-heuristic, etc. He is an Associate Editor of IEEE Transactions on Evolutionary Computation, and a guest editor of the Genetic Programming Evolvable Machine journal. He is a Fellow of Engineering New Zealand and an IEEE Senior Member.



Fangfang Zhang received the B.Sc. and M.Sc. degrees from Shenzhen University, Shenzhen, China, and the Ph.D. degree in Computer Science from Victoria University of Wellington, New Zealand, in 2014 and 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. Her research interests include evolutionary computation, hyper-heuristic learning/optimisation, job shop scheduling, surrogate, and multitask learning.

Hejiao Huang received her PhD degree in Computer Science from City University of Hong Kong in 2004. She is currently a professor in Harbin Institute of Technology, Shenzhen, China, and previously was an invited professor at INRIA, France. Her research interests include network security, cloud computing security, trustworthy computing, big data security, formal methods for system design and wireless networks.

Chonglin Gu received PhD degree in computer science and technology from Harbin Institute of Technology, Shenzhen in 2018. After that, he has been a postdoctoral fellow in the Chinese University of Hong Kong, Shenzhen, China. He is currently an assistant professor in the school of computer science and technology in Harbin Institute of Technology, Shenzhen. His research interests include cloud computing, especially algorithm design and system implementation.



Mengjie Zhang received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, and the Ph.D. 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 optimisation, job shop scheduling. Prof. Zhang is a Fellow of Royal Society of New Zealand, a Fellow of Engineering New Zealand, a Fellow of IEEE, an IEEE CIS Distinguished Lecturer.