Dual-tree Genetic Programming for Deadline-Constrained Dynamic Workflow Scheduling in Cloud

Yifan Yang[1], Gang Chen[1], Hui Ma[1], and Mengjie Zhang[1]

School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand
{yifan.yang, aaron.chen, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Dynamic workflow scheduling (DWS) aims to allocate abundant cloud resources to process a large number of heterogeneous workflows in order to minimize total operation cost and the penalty for violating deadline constraints. Instead of using manually designed heuristics that cannot work effectively across different problem instances, we develop a new Genetic Programming Hyper-Heuristic (GPHH) algorithm to automatically design scheduling heuristics for a newly formulated deadline-constrained dynamic workflow scheduling in cloud (DCDWSC) problem. Different from previous works, our GPHH algorithm can design a pair of rules for Virtual Machine selection and task selection. A new dual-tree representation is proposed to jointly evolve the rule pair, enabling the algorithm to effectively control the inter-dependencies of the two rules. Experimental results show that our new algorithm can significantly outperform three baseline algorithms on a wide range of testing scenarios. In addition, using GPHH to design the scheduling heuristic with two rules is more effective than using one rule (e.g., the VM selection rule) alone for DCDWSC.

Keywords: Dynamic workflow scheduling · Genetic programming hyper-heuristic · Cloud computing · Deadline constraint

1 Introduction

Many organizations are increasingly relying on cloud computing to process their workflows due to abundant heterogeneous computing resources and secure data storage in cloud [2]. For example, MetService in New Zealand uses multiple cloud resources to process its workflows for daily weather forecasting. Each *workflow* consists of a set of *tasks* with sophisticated *inter-dependencies* [13] and can be provided to numerous customers [2] based on pre-determined *Service Level Agreements* (SLAs). Effective methods are needed to help organisations to decide proper cloud resources for processing workflows to minimize the total cloud resource rental costs and SLA penalties [1, 17, 22]. In this paper, we use *brokers* to refer the agents who provide workflows as services in cloud.

For many brokers in cloud, *workflow scheduling* is a vital issue for them to lease and allocate suitable cloud computing resources (i.e., *Virtual Machines* or VMs) to execute a series of dynamically requested workflows in order to achieve important objectives, such as to minimize the total makespan and cost [16]. Many previous studies proposed various scheduling heuristics to tackle such workflow scheduling problems on the fly. However, manual design of scheduling heuristics demands for extensive human labor and domain expertise [7, 15, 21]. Further, manually designed heuristics can quickly lose effectiveness due to the increasing variety of cloud computing resources and workflow workloads. Furthermore, existing heuristics did not consider penalties resulted from violations of deadline constraints defined by SLA.

Hyper-heuristic techniques can automatically design a wide variety of heuristics and have been extensively utilized to solve diverse combinatorial optimization problems in recent years [11, 14, 25]. Particularly, a few research works recently developed *Genetic Programming Hyper-Heuristic* (GPHH) algorithms to tackle *dynamic workflow scheduling* (DWS) problems successfully. In [5, 6, 22], GPHH is explored to evolve a single rule/heuristic to select appropriate VM instances to process each workflow task. However, due to the highly dynamic nature of workflow execution in cloud, using VM selection rule alone is often insufficient (see Subsection 3.1 and Fig. 1 for detail). Hence, [20] developed a cooperative coevolution GP (CCGP) approach to evolve a pair of task prioritizing rule and VM selection rule. Nevertheless, this method was designed to execute a single workflow without explicitly controlling the inter-dependencies between the two evolved rules. Effective methods are needed to cooperatively generate two rules to schedule a sequence of workflows dynamically arriving the cloud.

The goal of this paper is to develop a new dual-tree GP (DTGP) algorithm to jointly evolve *VM selection rules* (VMSRs) and *task selection rules* (TSRs) to effectively solve the deadline-constrained dynamic workflow scheduling in cloud (DCDWSC) problem. Three major contributions have been achieved:

1. We formally model the DCDWSC problem and demonstrate its practical importance. Different from existing problems, the new problem model considered for the first time the possibility of reordering pending tasks in VM queues to reduce the total cost involved in executing multiple heterogeneous workflows.
2. We develop a DTGP algorithm that uses a new dual-tree representation to effectively support the joint evolution of VMSRs and TSRs and explicitly control their inter-dependencies. Multiple new terminal types (i.e., features used for making scheduling decisions) have also been proposed to facilitate the design of effective VMSRs and TSRs.
3. We conduct extensive experimental evaluation on diverse DCDWSC problem scenarios with a variety of heterogeneous workflows to verify the effectiveness of DTGP. DTGP is experimentally shown to significantly outperform several existing approaches.

2 Related Work

Workflow scheduling aims to allocate cloud resources to process all tasks of one or multiple workflows [16]. All the previously studied workflow scheduling problems are either *static* or *dynamic*.

Most of the static problems concern mainly about scheduling a single workflow [4, 15]. Dynamic resource provisioning is often neglected in the problem formulation. In view of the highly dynamic cloud computing environment, the research community is starting to pay more attention to the DWS problems [1,8,19] that consider either dynamic workflow arriving time [1, 19] or dynamic resource provisioning [8] based on pre-determined workflow patterns. In fact, existing DWS problems rarely handle multiple heterogeneous workflows simultaneously. Although some studies [6, 9] considered several different workflows patterns, a common assumption is to handle each workflow pattern one at a time. As far as we know, no existing studies considered the general and realistic problem for brokers to process a series of dynamically requested workflows with previously unknown patterns.

Most of existing research works focused on VM selection [5, 8]. In this paper, we study the DWS problem with deadline constraints that aims to maximize the total profit. On the one hand, it is challenging to decide the number and type of VM instances to rent. To address this issue, we use GPHH to evolve a *VM selection* rule to select VMs to process each task in order to balance the VM rental fees and deadline penalties. On the other hand, it is critical to determine suitable orders to process all tasks pending for execution on a VM instance. Therefore, we further use GPHH to evolve a *task selection* rule to control the processing order of pending tasks so as to reduce deadline violations and shorten the workflow makespan. To the best of our knowledge, this is the first work to jointly consider VM selection and task selection for DWS.

3 Problem Description

3.1 Problem Overview

We assume that workflow requests arrive at a cloud data center dynamically over time. There are a fixed number of VM types available at the data center. An unlimited number of instances can be rented with respect to each VM type. Every workflow consists of a number of tasks and has a workflow pattern and size (see Fig. 1), which are unknown before arrival. Any task can be allocated to either an existing VM instance or any newly leased instance.

Fig. 1 illustrates how a *scheduling heuristic* is used to schedule the execution of workflow tasks in the cloud. Different from several existing works [6, 20, 22], the scheduling heuristic in this paper is composed of a VMSR and a TSR. They are used to jointly support two interdependent scheduling decisions, which are highlighted as *VM selection* and *task selection* in Fig. 1, respectively.

Whenever one or multiple workflows arrive at the cloud, all the *ready tasks* within these workflows will be identified. For each ready task, the VMSR selects

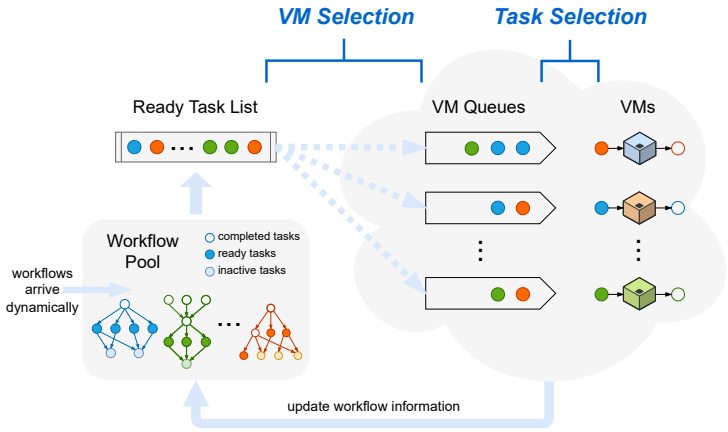


Fig. 1. The procedure of using a heuristic to schedule dynamic workflow execution.

from all candidate VM instances the instance with the highest-priority to process it. We do not only consider existing leased VM instances but also VM instances of any type that can be further leased for workflow execution. In other words, when a task is assigned to a VM instance that was not leased previously, we will lease the VM instance immediately. Upon allocating a task to a VM instance, if the VM instance is idle and has no pending task in its VM Queue, it immediately executes the allocated task; otherwise, the task is added into the instance's VM queue.

Whenever a VM instance completes its execution of one task, the TSR will be activated to select a pending task with the highest priority in its VM queue to be executed next. After a task is processed, some of its successor tasks in the same workflow will become ready. They will be subsequently allocated to either existing VM instances or newly leased VM instances. The above process will be repeated until all tasks of all workflows are processed. Afterwards, the VM rental fees in eq. (9) and the deadline violation penalties in eq. (11) will be computed to quantify the performance of the workflow scheduling process.

3.2 Formulation

The broker has no access to the size and pattern of any workflow before its arrival. The broker will also compensate its users if the execution of any workflow violates its deadline constraint specified in the respective SLA.

We consider a collection of dynamically arriving workflows for the DWS problem, denoted as $\mathcal{W} = \{W_1, W_2, ..., W_m\}$. A workflow W_i is defined as:

$$W_i = (DAG_i, AT_i, NOR_i, DL_i, RDL_i) \tag{1}$$

where DAG_i capture the workflow pattern, AT_i denotes the arriving time of the workflow, NOR_i is the total number of unassigned tasks remaining in the

workflow, and DL_i denotes the deadline of the workflow. Additionally, RDL_i is the remaining time before the deadline is due, which is calculated by

$$RDL_i = DL_i - current\_time \tag{2}$$

These workflows will be processed by a time varying set of VM instances, denoted as $\mathcal{V} = \{V_1, V_2, ..., V_n\}$. A VM instance V_k is defined as a tuple below:

$$V_k = (TYPE_k, NIQ_k, TIQ_k, VMR_k) \tag{3}$$

where $TYPE_k = (CU_k, MEM_k PRICE_k)$ denotes its VM type. CU_k is the compute unit (i.e., computation capacity), MEM_K is the memory size, and $PRICE_k$ is the hourly rental fee charged by cloud providers. NIQ_k refers to the number of pending tasks in the VM Queue of V_k, TIQ_k is the total execution time of all pending tasks in the queue, and VMR_k is the remaining lease period of V_k of its current lease.

Given the set of all tasks $\{t_{i1}, t_{i2}, ..., t_{iq}\}$ that belong to a workflow W_i, each task $t_{ij} = (NOC_{ij}, TS_{ij})$. NOC_{ij} is the number of its successor tasks. TS_{ij} refers to its size, i.e., the execution time of t_{ij} on a VM instance with the compute unit $CU = 1$. Accordingly, the actual execution time of t_{ij} on V_k is

$$ET_{ij}^k = \frac{TS_{ij}}{CU_k} \tag{4}$$

Furthermore, we can obtain the time RT_{ij} when the task t_{ij} is assigned to an VM instance:

$$RT_{ij} = \max_{z \in pred(t_{ij})} \{FT_z\} \tag{5}$$

where FT_z refer to the finish time of its predecessors. Eq. (5) implies that t_{ij} is assigned immediately to a VM instance when all its predecessor tasks are completed. Meanwhile, the ready time of the entry task is equal to the arrival time of its workflow, i.e., RT_{i1}=AT_i.

Let ST_{ij}^k be the time that the VM instance V_k starts to execute t_{ij}. The waiting time period WT_{ij}^k of t_{ij} and the finish time FT_{ij}^k of t_{ij} on V_k are defined as:

$$WT_{ij}^k = ST_{ij}^k - RT_{ij} \tag{6}$$

$$FT_{ij}^k = ST_{ij} + ET_{ij}^k \tag{7}$$

Through processing all workflows, we aim to minimize the *total cost* incurred for workflow execution, which consists of both *VM rental fees* and *workflow deadline penalties*, as formulated below

$$TotalCost = \sum_{k \in LVMS} RentFee_k + \sum_{i \in \mathcal{W}} Penalty_i \tag{8}$$

where *LVMS* is the set of leased VM instances. $\mathcal{W}$ is the set of all workflows to be executed within a given period of time. $RentFee_k$ denotes the rental fee of the VM instance V_k, and $Penalty_i$ denotes the deadline violation penalty of the workflow W_i, as defined below.

1. $RentFee_k$**:** We use the prevailing hourly-based cost model supported by the global cloud market [1, 7, 18]. The cost of renting any VM instance V_k is calculated by

$$RentFee_k = PRICE_k \times \left\lceil \frac{FT_{t^k_{last}} - ST_{t^k_{first}}}{3600} \right\rceil \tag{9}$$

where t^k_{first} and t^k_{last} are the first task and the last task executed on V_k. Thus, the numerator in eq. (9) gives the total time period measured in seconds between the start time of t^k_{first} and the finish time of t^k_{last}. The denominator and the ceiling function in eq. (9) together convert this time period into the total number of leased hours.

2. $Penalty_i$**:** We first define the deadline of a workflow, denoted as $Deadline_i$, below:

$$Deadline_i = AT_i + \xi \times MinMakespan_i \tag{10}$$

where AT_i represents the arrival time of the workflow W_i, and ξ is a *relaxation coefficient* [1]. $MinMakespan_i$ refers to the theoretical shortest completion time of workflow W_i by executing all of its tasks on the fastest VM instances without any delay. Whenever $Deadline_i$ is violated due to delay in executing workflow W_i, penalties will be incurred as determined below:

$$Penalty_i = \delta \max\{0, AT_i + Makespan_i - Deadline_i\} \tag{11}$$

where δ is a *penalty coefficient* [23]. The smaller the value of δ, the greater the tolerance for violating the deadline.

4 Algorithm

To enable GPHH to generate effective rules for the DCDWSC problems, in this paper we propose two distinct sets of terminals to be utilized to design *VMSRs* and *TSRs*, respectively (see Subsection 4.1 for an introduction of all terminals used in these rules). Note that these sets are different from terminals that are used by existing works [6, 20, 24], which only focus on building VMSRs, since we include the features related to deadline constraint of workflows. Specifically, we propose multiple new terminals for VMSR and a new set of terminals for building TSRs. These terminals enable us to develop a new DTGP algorithm to simultaneously evolve VMSRs and TSRs for DCDWSC.

In line with Algorithm 1, details regarding the solution representation, initialization, fitness evaluation of DTGP are presented below.

4.1 Representation

In DTGP, both VMSR and TSR are represented as GP trees. Different from [20], a pair of GP trees, one for VMSR and one for TSR, jointly form a single individual in this paper. As illustrated in the left of Fig. 2, each GP tree is a syntax tree with one root node and multiple leaf nodes. The intermediate

Algorithm 1: DTGP Algorithm for DCDWSC

Input: Training instances, parameter settings
Output: The best scheduling rule consisting of a VMSR and a TSR
// Representation
Determine the terminal set and the function set
// Initialization
while $N < PopSize$ **do**
  Randomly initialize an individual
end
$gen \leftarrow 0$
while $gen < MaxGen$ **do**
  // Fitness Evaluation
  for *ind in Pop* **do**
    $fitness(ind) \leftarrow 0$
    for *i = 1 to EvalNumber* **do**
      $fitness(ind) \leftarrow fitness(ind) + objective(ind)$
    end
    $fitness(ind) \leftarrow fitness(ind)/EvalNumber$
  end
  // Evolution
  Generate new population by genetic operators
  $gen \leftarrow gen + 1$
end
Return the best individual/heuristic/rule

nodes, such as $+, -, \times, \div$, are called the *function* nodes. Every leaf node must be a *terminal* that extracts problem-dependent features from the DCDWSC problem, such as TS, NIQ and ET in Table 1.

Following many existing works [6,12,20], we consider $\{+, -, \times, \div, max, min\}$ as *function* nodes in the GP trees. We introduce new VMSR terminals as well as a new terminal set particularly designed for TSR. Table 1 summarizes the two terminal sets for building VMSRs and TSRs. Depending on the feature types captured by each terminal, all terminals are divided into task-related, VM-related, workflow-related, and problem-specific terminals (see Table 1).

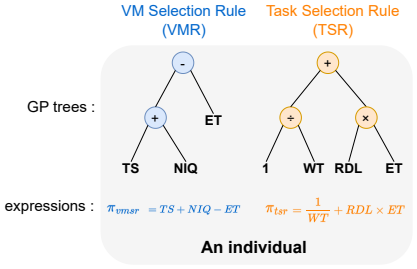
This paper considers five commonly used terminals for VMSR, i.e., TS, ET, CU, $PRICE$ and LFT [5,20,24]. Apart from that, we introduce three new VM-related terminals, TIQ, VMR, and NIQ. They enable VMSRs to evaluate the priority of processing any task on a VM instance based on the instance's current workload and remaining capacity. Two additional terminals, NOC and NOR, are also introduced to provide workflow-related information to VSMRs. RDL allows VSMRs to assign near-expire tasks to fast VM instances.

Among all terminals for designing TSRs, RWT is a time-varying terminal whose value can only be determined in the task selection phase instead of the VM selection phase. It enables a task with a long waiting time to have relatively high priority in the VM Queue. We also use two VM-related terminals TIQ

Table 1. The terminal set of VMSR and TSR.

	VMSR		TSR	
	Terminal	**Definition**	**Terminal**	**Definition**
task-related	TS	The size of a task	ET	The execution time of a task
	ET	The execution time of a task	RWT	The relative waiting time of a task in a VM queue
VM-related	CU	The compute unit of a VM	TIQ	Total execution time of all tasks in a VM queue
	$PRICE$	Price of renting a VM for one hour	NIQ	Number of tasks in a VM queue
	TIQ	Total execution time of all tasks in a VM queue		
	VMR	The remaining available time for a VM		
	LFT	The latest finish time of a task on a VM		
	NIQ	Number of tasks in a VM queue		
workflow-related	NOC	Number of successor tasks (children) of a task	NOC	Number of successor tasks (children) of a task
	NOR	Number of remaining tasks in a workflow	NOR	Number of remaining tasks in a workflow
problem-specific	RDL	Remaining deadline time of a workflow	RDL	Remaining deadline time of a workflow

and NIQ to capture the competition level among all tasks in the VM queue. NOC is expected to give high priority to those tasks with many successor tasks. NOR helps to shorten the completion time of those workflows with less pending tasks. RDL is important for satisfying deadline constraints. The usefulness of all the newly introduced terminals will be further analyzed experimentally in Subsection 5.6.



A decision situation for selecting a VM for $task_4$

Candidate VMs	TS	NIQ	ET	π_{vmsr}	Selected VM
VM_1	2	0	1	**1**	VM_1
VM_2	2	7	2	7	
VM_3	2	3	0.5	4.5	

A decision situation for selecting a Task for VM_3

Candidate Tasks	WT	RDL	ET	π_{tsr}	Selected Task
$task_1$	4	1.6	1	**1.85**	$task_1$
$task_2$	2	5	2.5	13	
$task_3$	1	2.8	0.5	2.4	

Fig. 2. Examples of how to use an individual to make scheduling decisions.

4.2 Initialization

The initial population is randomly generated by the widely used *Ramped half-and-half* approach [12] where half of the population is constructed by the *grow*

method (e.g., VMSR in Fig. 2) and half by the *full* method (e.g., TSR in Fig. 2). In the *grow* method, a GP tree grows by adding randomly selected function and terminal nodes to the tree until it reaches the initial depth limit (6 in our experiments). The *full* method randomly adds function nodes to the tree until it reaches the maximum tree depth.

4.3 Fitness Evaluation

Each evolved GP individual is evaluated on multiple problem instances (3 in our experiments) to determine its average performance in terms of eq. (8) as its fitness. Each problem instance involves a set of m heterogeneous workflows randomly sampled from multiple different workflow patterns (see Subsection 5.2 for more details).

Fig. 2 illustrates how to use a GP individual with a pair of VMSR and TSR to schedule workflow execution in the cloud. Specifically, it shows a decision situation that needs to select the optimal VM instance for $task_4$ from three candidate VM instances $\{VM_1, VM_2, VM_3\}$. We use VMSR to calculate the corresponding priority values of the three VM instances, i.e., $\{(2+0-1), (2+7-2), (2+3-0.5)\}$. Then, $task_4$ is allocated to the VM Queue of VM_1 which has the lowest priority value. Similarly, the priority values of all pending tasks on VM_3 are first calculated by TSR. $task_1$ with the lowest priority value is then selected for execution.

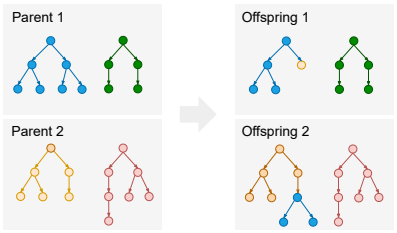


Fig. 3. Crossover operator.

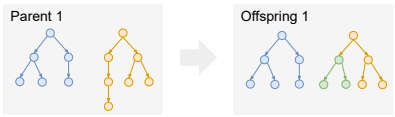


Fig. 4. Mutation operator.

4.4 Evolution

The evolution process relies on *crossover* and *mutation*, as demonstrated in Fig. 3 and Fig. 4. We use *single-point crossover* to process any two parent GP individuals, each represented as a rule pair. Specifically, crossover is applied to either the VMSRs or TSRs of the two individuals (or rule pairs) with a probability of 0.5. Similar to crossover, *mutation* is also applied to one randomly selected tree/rule of a GP individual. For the chosen tree/rule (either VSMR or TR), its sub-tree rooted at a randomly selected mutation point is replaced by a randomly generated new sub-tree.

5 Experiments

We conduct experimental evaluations to demonstrate the effectiveness of our proposed GPHH approach in generating high quality of heuristics for DCDWSC. This section begins by describing the parameter setting of GP and the simulation configuration. The effectiveness of the proposed approach is then verified experimentally, followed by terminal analysis and rule analysis.

5.1 Parameter Setting

Following [12], we set the population size of DTGP to 1024, the number of generations to 51, and the tournament size to 7. Crossover, mutation, and reproduction rates are 85%, 10%, and 5%, respectively. We also limit the initial depth of a GP tree to between 2 and 6, and its maximum depth to 10.

5.2 Simulation Configuration

A simulated cloud environment with five key components below is used to experimentally compare the performance of DTGP against several baseline approaches.

VM Types: The simulated cloud data center is equipped with 6 different VM types according to Amazon EC2[1], as summarized in Table 2. The number of instances of each VM type is unlimited.

Table 2. Configurations of 6 VM instances based on Amazon EC2

Instance name	vCPU	Memory	On-Demand hourly rate
m5.large	2	8 GiB	$0.096
m5.xlarge	4	16 GiB	$0.192
m5.2xlarge	8	32 GiB	$0.384
m5.4xlarge	16	64 GiB	$0.768
m5.8xlarge	32	128 GiB	$1.536
m5.12xlarge	48	192 GiB	$2.304

Workflow Set: Four popular workflow patterns[2] (i.e., CyberShake, Inspiral, Montage, and SIPHT) are employed for our experiments. Each is available with three different sizes (i.e., number of tasks in a workflow), including 30, 50, and 100. Using these workflows patterns, three scenarios (*Mix_Small*, *Mix_Medium* and *Mix_Large*) are created in Table 3.

Training scenario: The *Mix_Large* in Table 3 is utilized for training in GP-based approaches. Each GP individual will be evaluated on 3 problem instances randomly created from this scenario to calculate its fitness. Consequently, it takes 51×3 problem instances to run a GP-based algorithm till completion.

Testing scenario: All generated heuristics are tested on three *testing scenarios*, i.e., *Mix_Small*, *Mix_Medium* and *Mix_Large*, each containing 30 problem instances.

[1] https://aws.amazon.com/ec2/pricing/on-demand/

[2] https://confluence.pegasus.isi.edu/display/pegasus/Deprecated+Workflow+Generator

Table 3. Workflow patterns contained in three workflow sets.

Scenario	Workflow set	# of Workflows
Mix_Small	CyberShake_30, Inspiral_30, Montage_25, Sipht_30	30
Mix_Medium	CyberShake_30, Inspiral_30, Montage_25, Sipht_30 , CyberShake_50, Inspiral_50, Montage_50, Sipht_60	30
Mix_Large	CyberShake_30, Inspiral_30, Montage_25, Sipht_30 , CyberShake_50, Inspiral_50, Montage_50, Sipht_60, CyberShake_100, Inspiral_100, Montage_100, Sipht_100	30

Request generation: Workflow requests arrive at the cloud data center over time following a Poisson distribution with $\lambda = 0.01$ [10]. The penalty coefficient in eq. (11) is $\delta = \$0.24/h$ according to [23]. Moreover, the deadline relaxation coefficient in eq. (10) is set to $\xi \in \{1, 12, 24, 36\}$, where a larger ξ implies more relaxed deadline which can be fulfilled by using relatively cheaper VMs.

5.3 Baseline Algorithms

This paper compares three baseline algorithms listed below, including two GP-based algorithms [20,22] and one well-known heuristic approach [3,15]. All GP-based algorithms will run independently for 30 times using the same set of problem instances. The final performance of any GP-based algorithm is then calculated as the average total cost achieved by the 30 best scheduling heuristics obtained from each of the 30 runs on all testing scenarios.

- *HEFT-FCFS* [3,15] uses HEFT for VM selection and FCFS for task selection.
- *SGP* [5] is a GPHH approach that can evolve VMSRs for DWS.
- *CCGP* [20] is a cooperative coevolution GPHH approach that evolves a combination of one VMSR and one TSR via two evolutionary sub-populations.

5.4 Performance Comparison

The test performance of all algorithms on three scenarios (see Table 3) with four deadline relaxation coefficients ($\xi = 1, 12, 24, 36$) is summarized in Table 4, which records the best and mean total costs in (8) across 30 independent runs. To identify whether there is a statistically significant difference among all competing algorithms, a Wilcoxon test with a significance level of 0.05 is performed between each pair of algorithms. All statistically significant results are indicated as "+", "−" or "=" in Table 4. The optimal value in each row is also bolded.

Compared to other baselines, DTGP achieved the lowest overall costs on most of the testing scenarios and performed effectively under both tight ($\xi = 1$) and loose ($\xi = 24, 36$) deadlines because it can prioritize tasks with high overdue risks through reordering all pending tasks in the VM queues. Furthermore, for GP-based algorithms, the total cost decreases upon increasing ξ since they allow workflows with loose deadlines to be processed on cheaper VM instances.

Interestingly, SGP outperforms CCGP and DTGP when $\xi = 12$. We notice that when the deadline is at a moderate level, tasks are normally processed in

Table 4. The best and mean(standard deviation) objective values of 4 algorithms on 12 testing scenarios across 30 independent runs.

Scenarios		HEFT-FCFS		SGP		CCGP		DTGP	
		best	*mean(std.)*	*best*	*mean(std.)*	*best*	*mean(std.)*	*best*	*mean(std.)*
$\xi = 1$	S	87.55	103.58(9.12)	58.12	60.84(2.25)(+)	57.51	60.94(3.07)(+)(=)	57.26	**59.65(1.68)(+)(+)(+)**
	M	145.15	164.20(10.07)	70.02	73.72(3.31)(+)	69.64	73.48(3.17)(+)(=)	69.01	**71.74(2.03)(+)(+)(+)**
	L	246.53	283.93(17.98)	86.36	91.10(4.45)(+)	85.04	91.20(4.8)(+)(=)	84.65	**88.76(3.45)(+)(+)(+)**
$\xi = 12$	S	91.39	101.50(6.57)	26.34	**31.91(3.59)(+)**	27.41	34.27(3.62)(+)(-)	27.24	33.71(4.70)(+)(-)(+)
	M	152.06	167.94(11.34)	39.01	**45.26(4.23)(+)**	39.08	49.51(5.4)(+)(-)	39.06	48.50(7.05)(+)(-)(+)
	L	248.06	283.47(22.43)	54.91	**65.30(5.84)(+)**	56.74	70.4(7.85)(+)(-)	57.29	69.30(10.63)(+)(-)(+)
$\xi = 24$	S	89.86	104.68(8.78)	25.74	28.49(1.53)(+)	24.34	29.27(1.96)(+)(-)	23.62	**26.16(1.79)(+)(+)(+)**
	M	139.78	167.14(13.28)	37.23	42.13(2.98)(+)	35.34	44.01(3.08)(+)(-)	34.91	**38.46(2.72)(+)(+)(+)**
	L	243.46	272.54(20.32)	52.84	62.51(4.93)(+)	51.8	65.60(4.84)(+)(-)	50.76	**57.05(5.34)(+)(+)(+)**
$\xi = 36$	S	90.62	103.45(6.93)	24.65	62.51(4.93)(+)	23.36	25.97(2.26)(+)(+)	23.09	**23.76(0.53)(+)(+)(+)**
	M	148.22	167.45(10.44)	36.63	38.95(1.70)(+)	33.95	38.35(3.55)(+)(+)	32.98	**34.53(0.96)(+)(+)(+)**
	L	246.53	270.85(17.29)	52.33	56.85(2.44)(+)	50.12	58.03(7.51)(+)(-)	48.88	**51.27(1.84)(+)(+)(+)**

* (+), (-) or (=) indicates that the matching result is significantly better, worse, or equivalent to its counterpart.

a FCFS order on any VM instances [3, 22]. Hence, without evolving TSRs, SGP can concentrate fully on evolving more effective VMSRs with a much smaller search space than that of CCGP and DTGP. Comparing CCGP and DTGP, the results in Table 4 clearly indicate that simultaneously evolving VMSR and TSR as a dual-tree is more effective than evolving them separately in two sub-populations. By using the best GP tree in one sub-population (e.g., the best VMSR) to evaluate the fitness of all GP trees in another sub-population (e.g., TSRs), CCGP does not explore all potentially useful combinations of VMSRs and TSRs from both sub-populations.

5.5 Ablation Study

To demonstrate the necessity and effectiveness of jointly using both VSMR and TSR, we compare the performance of a rule pair designed by DTGP with the performance achieved by using only VMSR in the same rule pair on 12 testing scenarios. The observed performance difference is captured by a percentage metric defined in eq. (12).

$$\frac{fitness(VMSR) - fitness(VMSR, TSR)}{fitness(VMSR, TSR)} \times 100\% \tag{12}$$

Table 5. Percentage increase in total cost when using VMSRs alone.

	Mix_Small	*Mix_Medium*	*Mix_Large*
$\xi = 1$	6.31%	6.96%	6.96%
$\xi = 12$	53.00%	43.45%	32.47%
$\xi = 24$	87.10%	75.12%	59.18%
$\xi = 36$	123.40%	103.16%	80.35%

Table 5 shows the percentage increase in total costs when using the VMSR of a rule pair evolved by DTGP alone. The results demonstrate that using two rules to schedule workflows is substantially better than using VMSR only on all testing scenarios. TSR therefore plays an essential role in solving the DCDWSC problem.

5.6 Terminal Analysis

We further analyze the distribution of terminal nodes among the best 30 rule pairs generated by DTGP in 30 runs to verify whether the newly proposed terminals in Subsection 4.1 are effective. Specifically, VMR, NIQ, NOC, NOR and RDL are newly developed terminals for designing VMSRs. We calculate the percentage of the number of each terminal with respect to the total number of terminals in a rule, and report the average percentage among the 30 rules in Fig. 5 and Fig. 6.

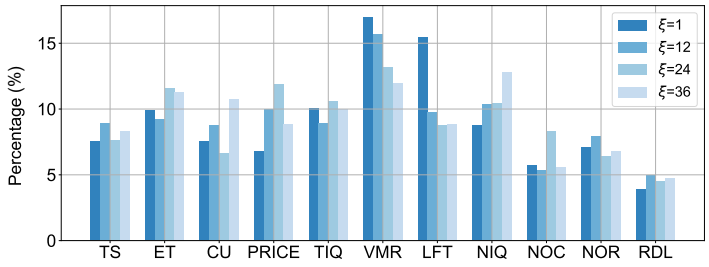


Fig. 5. Terminal statistic of VMSRs.

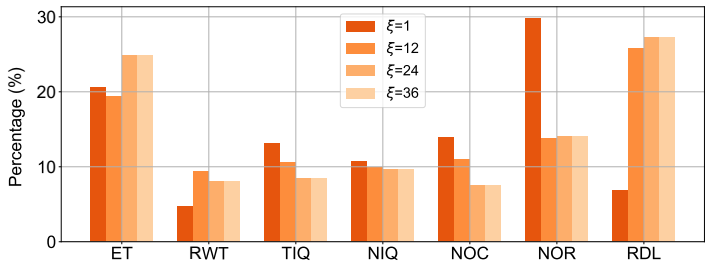


Fig. 6. Terminal statistic of TSRs.

In Fig. 5, the top three terminal types used in VMSRs are VMR, LFT and NIQ. They capture the real-time information of the VM. This is because VM-related information has a significant impact on the performance of VMSRs. Fig. 6 shows that ET, NOR and RDL are the top three terminal types in TSRs. Clearly, with tight deadlines (i.e., $\xi = 1$), the number of unassigned tasks remaining in a workflow (NOR) has a strong influence on the TSR. The remaining time before the deadline of a workflow (RDL) also affects strongly the processing order of those tasks waiting at a VM queue.

6 Conclusions

In this paper, we investigated the DCDWSC problem where a series of heterogeneous workflows can arrive dynamically over time with varied deadline constraints. To address this problem, we proposed the DTGP algorithm to jointly design a pair of VMSR and TSR. Both VMSR and TSR are supported by newly developed terminals. As far as we know, TSR has never been used in previous studies. Experimental results confirm that DTGP can outperform several competing algorithms under both tight and loose deadlines. Moreover, we found experimentally that better performance can be achieved by using both VSMR and TSR, instead of using VMSR alone. Evolving VSMRs and TSRs in the form of dual-trees was proved to be more effective than evolving them in separate sub-populations.

In the future, effective recombination methods can be further developed to improve the performance of DTGP. The influence of deadline penalty factor on the formation of the two rules can also be analyzed.

References

1. Arabnejad, V., Bubendorfer, K., Ng, B.: Dynamic multi-workflow scheduling: A deadline and cost-aware approach for commercial clouds. Futur. Gener. Comp. Syst. **100**, 98–108 (2019)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., et al.: Above the clouds: A berkeley view of cloud computing. Tech. rep. (2009)
3. Chen, W., Deelman, E.: Workflowsim: a toolkit for simulating scientific workflows in distributed environments. In: 2012 IEEE 8th international conference on E-science. pp. 1–8. IEEE (2012)
4. Djigal, H., Feng, J., Lu, J., Ge, J.: Ippts: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **32**(5), 1057–1071 (2020)
5. Escott, K.R., Ma, H., Chen, G.: Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud. In: International Conference on Database and Expert Systems Applications. pp. 76–90. Springer (2020)
6. Escott, K.R., Ma, H., Chen, G.: A genetic programming hyper-heuristic approach to design high-level heuristics for dynamic workflow scheduling in cloud. In: 2020 IEEE Symposium Series on Computational Intelligence. pp. 3141–3148. IEEE (2020)
7. Faragardi, H.R., Saleh Sedghpour, M.R., Fazliahmadi, S., Fahringer, T., Rasouli, N.: Grp-heft: a budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds. IEEE Trans. Parallel Distrib. Syst. **31**(6), 1239–1254 (2020)
8. Ismayilov, G., Topcuoglu, H.R.: Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Futur. Gener. Comp. Syst. **102**, 307–322 (2020)
9. Kenari, A.R., Shamsi, M.: A hyper-heuristic selector algorithm for cloud computing scheduling based on workflow features. OPSEARCH **58**(4), 852–868 (2021)

10. Liu, J., Ren, J., Dai, W., Zhang, D., Zhou, P., Zhang, Y., Min, G., Najjari, N.: Online multi-workflow scheduling under uncertain task execution time in iaas clouds. IEEE Trans. Cloud Comput. **9**(3), 1180–1194 (2019)
11. Liu, Y., Mei, Y., Zhang, M., Zhang, Z.: A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem. Evol. Comput. **28**(2), 289–316 (2020)
12. O'Neill, M.: Riccardo poli, william b. langdon, nicholas f. mcphee: a field guide to genetic programming (2009)
13. Rizvi, N., Dharavath, R., Wang, L., Basava, A.: A workflow scheduling approach with modified fuzzy adaptive genetic algorithm in iaas clouds. IEEE Trans. Serv. Comput. pp. 1–1 (2022). https://doi.org/10.1109/TSC.2022.3174112
14. Tan, B., Ma, H., Mei, Y., Zhang, M.: A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. IEEE Trans. Cloud Comput. **10**(3), 1500–1514 (2022). https://doi.org/10.1109/TCC.2020.3026338
15. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
16. Versluis, L., Iosup, A.: A survey of domains in workflow scheduling in computing infrastructures: Community and keyword analysis, emerging trends, and taxonomies. Futur. Gener. Comp. Syst. **123**, 156–177 (2021)
17. Wang, Z.J., Zhan, Z.H., Yu, W.J., Lin, Y., Zhang, J., Gu, T.L., Zhang, J.: Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. IEEE T. Cybern. **50**(6), 2715–2729 (2020)
18. Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., Wen, J.: Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Trans. Parallel Distrib. Syst. **28**(12), 3401–3412 (2017)
19. Xiao, J.P., Hu, X.M., Chen, W.N.: Dynamic cloud workflow scheduling with a heuristic-based encoding genetic algorithm. In: International Conference on Neural Information Processing. pp. 38–49. Springer (2020)
20. Xiao, Q.z., Zhong, J., Feng, L., Luo, L., Lv, J.: A cooperative coevolution hyper-heuristic framework for workflow scheduling problem. IEEE Trans. Serv. Comput. **15**(1), 150–163 (2022)
21. Xie, Y., Gui, F.X., Wang, W.J., Chien, C.F.: A two-stage multi-population genetic algorithm with heuristics for workflow scheduling in heterogeneous distributed computing environments. IEEE Trans. Cloud Comput. pp. 1–1 (2021). https://doi.org/10.1109/TCC.2021.3137881
22. Yang, Y., Chen, G., Ma, H., Zhang, M., Huang, V.: Budget and sla aware dynamic workflow scheduling in cloud computing with heterogeneous resources. In: 2021 IEEE Congress on Evolutionary Computation. pp. 2141–2148 (2021)
23. Youn, C.H., Chen, M., Dazzi, P.: Cloud broker and cloudlet for workflow scheduling. Springer (2017)
24. Yu, Y., Feng, Y., Ma, H., Chen, A., Wang, C.: Achieving flexible scheduling of heterogeneous workflows in cloud through a genetic programming based approach. In: 2019 IEEE Congress on Evolutionary Computation. pp. 3102–3109. IEEE (2019)
25. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling. IEEE Trans. Evol. Comput. **25**(3), 552–566 (2021)