A model-driven machine learning approach to dynamic multi-workflow scheduling

Yifan Yang[1], Hui Ma[1,*], Gang Chen[1] and Sven Hartmann[2]

[1]Victoria University of Wellington, Wellington, New Zealand
[2]Clausthal University of Technology, Clausthal-Zellerfeld, Germany

Abstract

Machine learning is getting more and more popular in solving dynamic combinatorial optimization problems. With the popularity of cloud computing, machine learning has been applied in solving dynamic multi-workflow scheduling, a challenging combinatorial optimization problem that involves many features of the problem, i.e., workflow patterns, tasks, virtual machine capacities, and cost. Existing machine learning approaches to dynamic combinatorial optimization problems use randomly generated training instances to train optimization rules or heuristics, without paying attention to the knowledge of the problem domain data. In this paper, we propose a model-driven machine learning approach to dynamic multi-workflow scheduling. In particular, we propose a conceptual database schema that can be used to model the information of the dynamic multi-workflow scheduling problem. We further propose a new approach to design training instances for dynamic multi-workflow scheduling by incorporating important attributes of the problem into the database schema. We evaluate our proposed approach using a case study with real-world data. The results of the evaluation demonstrate that our proposed model-driven machine learning approach can effectively solve dynamic workflow scheduling problems.

Keywords

machine learning, dynamic workflow scheduling, data modeling

CEUR-WS.org/Vol-3618/forum_paper_26.pdf

1. Introduction

In the era of big data, more and more data-intensive applications are used in various domains to solve complex data-intensive problems. To effectively solve such problems we need to design not only effective machine learning algorithms but also the training datasets to capture the essential requirements of the problem domain. However, with the increasing complexity of combinatorial optimization problems, designing training datasets for a problem is getting more and more challenging, i.e., to be representative of the problem domain and robust to unseen problem instances while at the same time without hurting the efficiency of the training process. For example, combinatorial optimization is a category of optimization where the size and relationships of data are massively increasing due to the escalated problem complexity [1, 2].

A typical example of combinatorial problems is the dynamic workflow scheduling problem in cloud computing where workflow tasks are scheduled to cloud resources in the best possible

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal

*Corresponding author.
hui.ma@ecs.vuw.ac.nz (H. Ma); sven.hartmann@tu-clausthal.de (S. Hartmann)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

way with some objectives, to minimize the makespan and/or response time. Cloud users often request multiple workflows to be scheduled for execution on heterogeneous cloud resources. This gives rise to the *dynamic multi-workflow scheduling (DMWS)* problem. To solve such a combinatoral optimization problem, we need to use not only data about tasks and cloud resources (e.g., VMs), but also relationships among these tasks and workflows, because they can significantly affect the response time and the overall cost.

Dynamic multi-workflow scheduling is an NP-hard problem [3]. A major challenge in practice is that solutions must be determined in real-time. For the DMWS, workflow scheduling must be decided in real-time. Therefore, heuristics need to be used. The manual design of heuristics is very time consuming and requires domain experts who have good domain knowledge. *Genetic programming hyper heuristic (GPHH)*, a machine learning method, is an effective approach to automatically design heuristics that does not require domain experts [4]. However, datasets are needed to train heuristics so that they can solve the problem effectively in real-time.

To efficiently search for heuristics for dynamic multi-workflow scheduling, the quality of datasets used for training heuristics plays a crucial role to make the heuristics effective in practice such as robustness to unseen problem instances [5]. While it is a common practice to use historical datasets as training instances, they are not efficient for evaluating heuristics that involve expensive evaluations over all the historical data [6]. To avoid expensive training while at the same time to train heuristics that are robust to unseen problem instances, it is important to carefully design the training instances. Current approaches of designing training instances are ad hoc, and are not clearly described. Most of existing work use randomly generated training instance in the training process [7], without any attention to the knowledge of the datasets given. Further, dynamic workflow scheduling is a dynamic combinatorial optimization problem for which different states of the problem domain will be used during the training process, including resource scheduling requests and resource status, which are changing dynamically. Meanwhile, to train heuristics that are effective and robust in solving dynamic multi-workflow scheduling problems in real time, it is inefficient and unnecessary to use all the possible states of the requests and data centers.

A conceptual model of training datasets, consisting of the data and relationships between different data, would help machine learning to be systematic and effective. A conceptual database schema consists of entities and relationships. Designing training data is actually retrieving, or sampling, datasets from database of historical data. This makes conceptual model attractive for the machine learning domain since it provides a high-level view of the datasets. However, there is no data model in the literature that can be used to support machine learning processes to learn heuristics effectively and efficiently when there is a large number of requests and the size of the workflow is extremely large.

To take advantage of conceptual models different views of the datasets can be retrieved as training datasets to efficiently evaluate and train heuristics [8]. Adequate modeling of historical data and training data plays an important role to design a high-quality training dataset for machine learning. However, despite the popularity of machine learning for combinatorial optimization, there is limited research work on data modeling for machine learning. A data model, GR4ML,is proposed in [9] for machine learning focusing on expressing machine learning requirements and solution design. In practice, datasets are often designed in an ad hoc way or based on best practice recommendations [10]. There are no common or well-defined approaches

to model training datasets and design training instances using data models.

The overall aim of this paper is to propose a model-driven machine learning approach to the dynamic multi-workflow scheduling problem. In particular, we propose a GPHH approach that trains heuristics using well defined training instances based on conceptual database schema. We expect that the heuristics learned from our machine learning approaches can schedule dynamic workflows effectively. The major contributions of this paper are the following:

- We propose a conceptual model that can be used for designing training datasets for effective learning heuristics of dynamic workflow scheduling.
- We present a model-driven GPHH approach to automatically learn heuristics that can be used to schedule multiple workflows in cloud data centers. In particular, our proposed GPHH approach will use training datasets generated using the conceptual model designed for DMWS.
- We evaluate the effectiveness of our proposed conceptual model in designing training datasets with a case study using benchmark data of workflow scheduling.

To the best of our knowledge, our proposed model-driven approach is the first attempt incorporating conceptual modeling into the process of machine learning for solving dynamic workflow scheduling and other combinatorial optimization problems. In particular our conceptual database schema provides a good overview of the structure of data involved in dynamic workflow scheduling problems, to support systematically the design of training instances for effectively learning heuristics that can be used to efficiently schedule tasks of numerous dynamically arriving workflows.

Organization. This paper is organized as follows. Section 2 briefly outlines current approaches for machine learning, in particular for dynamic multi-workflow scheduling problems from the literature, while in Section 3 we introduce dynamic workflow scheduling in more detail. In Section 4 we present a conceptual data schema for dynamic multi-workflow scheduling, and propose a GPHH approach to automatically design heuristics for the DMWS problem, with a model-driven approach for designing training instances. In Section 5 we report on a case study that we have conducted to evaluate our proposed model-driven machine learning approach. Finally, Section 6 gives conclusions and an outlook on future work.

2. Related work

As dynamic multi-workflow scheduling is known to be NP-hard, heuristics are widely used to ensure scalability and real-time applicability. Heuristics are priority functions for making decisions on cloud resources, e.g., VMs. IaaS cloud partial critical paths (IC-PCP) is proposed in [11] for scheduling computation requests in data centers. Heterogeneous earliest finish time (HEFT) is another manually designed heuristic proposed in [12]. However, manually designing such heuristics requires domain knowledge and is time-consuming. Also heuristics may be effective in some of problem instances but may not be effective in some other problem instances since only small set of domain attributes are considered.

Machine learning approaches have been proposed in the literature to automatically learn heuristics for dynamic workflow scheduling problems and other dynamic scheduling problems.

In [13, 14], GPHH is utilized for online service resource allocation in clouds. In [15], a GPHH approach is proposed to solve multi-cloud resource allocation problems. A training instance is used during the whole training process. In order to generate heuristics which are robust to unseen problem instances, in [16], a GPHH approach is proposed to use training instances that are randomly generated during the training process, to learn heuristics for the DWS problem. Similar approaches of generating training instances are used in [7], which proposes a GPHH approach to learn heuristics for the dynamic job shop scheduling problem. However, none of the above mentioned machine learning studies paid attention to the structure of problem attributes or data when designing training instances.

To understand machine leaning requirements of organizations, conceptual modelers have proposed different models. In [17], a conceptual modeling framework is proposed to systematically justify the needs of predictive analytics within an organizational context and to identify the requirements of predictive analytics design. To incorporate domain-specific questions, [18] adopts conceptual modeling methods to elicit analytical requirements, design machine learning solutions, and to ensure the alignment between analytic initiatives and business strategies, among others. The modeling framework consists of three complementary modeling views: business view, analytic design view, and data preparation view. However, as far as we are aware, there is no systematic way of designing training instances for machine learning in dynamic combinatorial optimisation problems, including dynamic workflow scheduling problems.

In summary, research on data modeling of training instances for machine learning of combinatorial optimization is still in its early stages. To the best of our knowledge, there is very limited work on conceptual modeling of machine learning data of dynamic multi-workflow scheduling, which motivate us to propose an effective design method of training instances by incorporating application-specific data models.

3. Preliminaries

SLA-aware dynamic multi-workflow scheduling (DWMS) has attracted rising interests in research and practice. Workflows arrive at a data center dynamically and is unknown beforehand, i.e., requests of scheduling workflows arrive on the fly and must be executed in real-time, to meet deadline constraints set in service-level agreements (SLA). The tasks of the workflows are scheduled to execute on a number of VM instances that are available in a cloud data center. Each VM instance is of a VM type, which defines its capacities. During the process, we need to make two types of decisions concurrently: *task allocation*, which allocates the tasks of the workflows to VMs, and *task sequencing*, which sequence the task on each individual VM instance.

Figure 1 shows an overview of DWMS. The overall goal is to satisfy the workflow scheduling requests in the best possible quality, e.g., minimizing the makespan of workflows, minimizing the costs of renting cloud resources, and minimizing SLA violation penalties. In particular, the SLA-aware dynamic multi-workflow scheduling problem focuses on scheduling a sequence of workflows to cloud resources so that the overall cost and SLA penalty are minimized. In the rest of the paper, DMWS refers the SLA-aware dynamic multi-workflow scheduling problem.

A workflow W arrives at a time with a deadline, can be represented as a directed acyclic graph (DAG), denoted by $G = (T, E)$. Herein, $T = \{t_i\}, i \in 1, \ldots, N$ is a set of nodes representing tasks

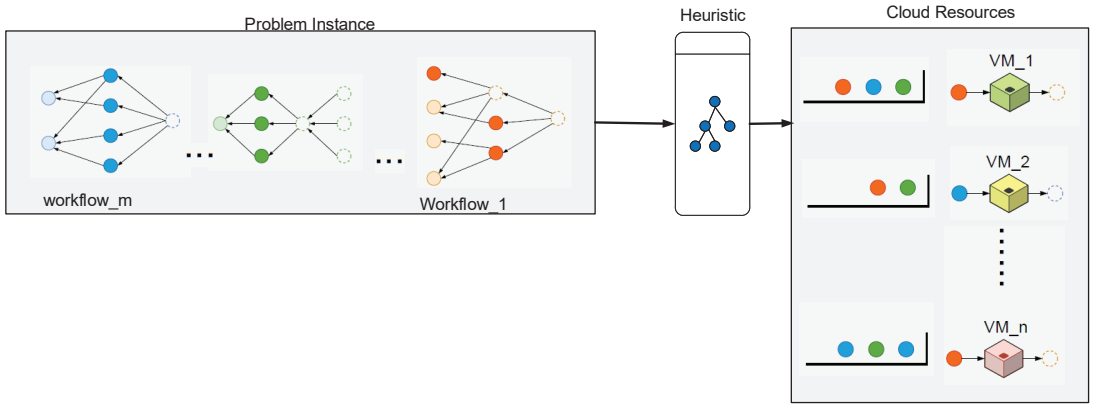


Figure 1: Dynamic multi-workflow scheduling in a cloud data center

in the workflow, and $E = \{e_{ij}\}$ is a set of directed edges where e_{ij} connects t_i with t_j. A task t_i is defined by its size s_i. Here, t_i is a parent task and t_j is a child task. Note that a child task can be executed only after all its parent tasks are completed. We use $Parent(t_i)$ to denote all immediate predecessors of task t_i. There might be multiple tasks allocated to the same VM instance to be executed. We use $Pre(t_i)$ to denote the previous task of t_i executed on the same VM.

In a data center, there are VM instances to perform scheduled workflow tasks. Each VM instance is of a VM type that is defined by its speed v_k, memory m_k, and cost c_k. Any task must be scheduled to a VM instance with sufficient capacity/resource, i.e., CPU and memory. Note that, one VM instance can only execute one task at any time, and each task can only be executed on a single VM instance. The time to process a task depends on the resource requirement s_i of the task, and the resource v_m that is scheduled. The more capacity of the resources allocated to the task the faster it is processed, and less probability to violate deadline constraints set in the SLA agreement. However, the more resources used for executing workflows, the more cost it will take. Different workflow schedules will lead to different time units for renting VMs, and then different makespan and costs.

For a given sequence workflows $\mathscr{W} = [W_1, \ldots, W_k]$, and set of available VM instances, the objective of the DMWS problem is to schedule workflow tasks for execution on VMs so as to minimize the total cost for workflow execution, consisting of *rental fees* (denoted as *RentFee*) incurred from resource provisioning and *deadline penalties* (denoted as *Penalty*) caused by violations, formulated by

$$\min TotalCost = \sum_{k \in \mathscr{V}} RentFee_k + \sum_{i:W_i \in \mathscr{W}} Penalty_i \tag{1}$$

where $\mathscr{V}$ refers to the set of rented VMs. Particularly, $RentFee_k$ is the rental fee of VM V_k, defined as the hourly-based fee for the rental time V_k between the time of processing of its first allocated

task $FT^k_{t_{last}}$ and the time of processing the last allocated task $ST^k_{t_{first}}$ calculated by

$$RentFee_k = PRICE_k \cdot \left\lceil \frac{FT^k_{t_{last}} - ST^k_{t_{first}}}{3600} \right\rceil \tag{2}$$

$Penalty_i$ is the penalty of workflow W_i, defined as the penalty fee paid for the portion beyond its deadline DL_i, calculated by

$$Penalty_i = \delta \cdot \max\{0, AT_i + Makespan_i - DL_i\} \tag{3}$$

where δ is a *penalty coefficient* [19]. A larger coefficient represents a lower tolerance for violating workflow deadline. Details of the problem definition can be found in [4].

4. Model-driven GPHH for dynamic multi-workflow scheduling

In this section we present our model-driven genetic programming hyper heuristic (GPHH) approach for the dynamic multi-workflow scheduling problem. To begin with we present a data model of the domain data for scheduling multiple dynamic workflows. Afterwards we describe our proposed GPHH approach using model-driven designed training instances.

Data model. To ensure the robustness of heuristics obtained by machine learning, we first model the data involved in the training of heuristics by GPHH. Figure 2 presents a conceptual data schema, denoted as ER-DMWS, for the problem of dynamic multi-workflow scheduling.

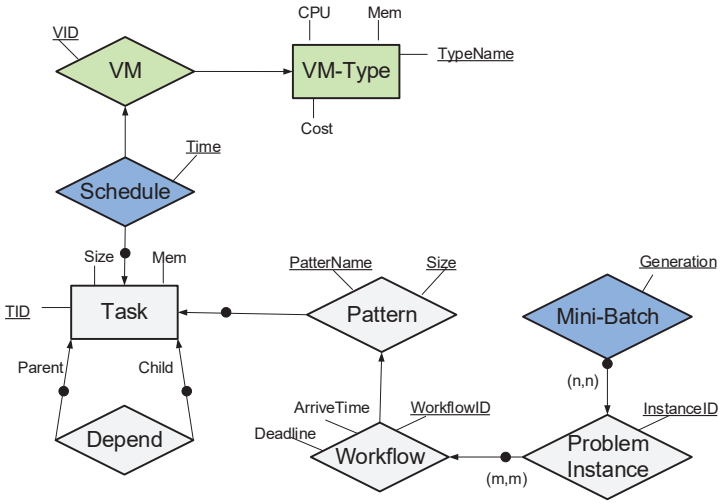


Figure 2: ER-DMWS - Data model for dynamic workflow scheduling training instances

We utilize the higher-order ER model (HERM) [20] which preserves the aggregation-based modeling principle of the basic ER model where a conceptual data schema consists of a set of object types. Object types are characterized by their attributes. Recall the definition of attributes

from [20]. Let $\mathcal{U}$ denote a countable set of simple attributes (called the *universe*) together with a type assignment $tp(A)$ that assigns to each attribute $A \in \mathcal{U}$ its data type $tp(A)$. Complex attributes may be obtained from simple ones by nesting. Let $\mathcal{A}$ denote the smallest superset of $\mathcal{U}$ such that

$$X(A_1, \ldots, A_n), X\{A\}, X[A], X\langle A\rangle, X_1(A_1) \oplus \cdots \oplus X_n(A_n), X(A_1 \to A_2) \in \mathcal{A}$$

holds whenever $A, A_1, \ldots, A_n \in \mathcal{A}$ holds, with labels $X, X_1, \ldots, X_n$ chosen from some fixed alphabet $\mathcal{L}$.

An important extension of HERM is that dynamic and complex data types are permitted in $\mathcal{A}$. The data type $tp(A)$ assigned to a (simple) attribute $A \in \mathcal{U}$ is some *simple data type*, like *INT*, *FLOAT*, *STRING*, *DATE*, or *TIME*, which are relevant features of dynamic workflow scheduling problems. It should be emphasized that complex data types offer additional opportunities for machine learning researchers but are not a must-use.

A *structured component* is a pair $\rho : C$ with a *role name* ρ and a component expression $C \in \mathcal{C}$. An *object type* O is a triple $(comp(O), attr(O), key(O))$ where $comp(O)$ is a finite set $\{\rho_1 : C_1, \ldots, \rho_n : C_n\}$ of structured components with pairwise different role names $\rho_1, \ldots, \rho_n$, $attr(O)$ is a finite set $\{A_1, \ldots, A_m\} \subseteq \mathcal{A}$ of complex attributes, and $key(O) \subseteq comp(O) \cup attr(O)$ is the primary key of O. An object type O is called an entity type if $comp(O)$ is the empty set, otherwise it is called a relationship type.

Example 1. *In the diagram of ER-DMWS in Figure 2, the entity types are displayed as rectangles while relationship types are displayed as diamonds. Let O be a relationship type and X_i a component of O, say in the form O_i or $p_i : O_i$. In the diagram this is displayed by an edge from O to O_i. The relationship type Depend in Figure 2 can be specified as follows:*

$$Depend = (\{Parent{:}\ Task, Child{:}\ Task\}, \{\}, \{Parent{:}\ Task, Child{:}\ Task\})$$

It has two structured components, each with different roles, no attributes, and its primary key includes both structured components. A cardinality constraint is an expression $card(O, X_i) = (a, b)$ where a is a natural number, and b is a natural number or ∞. For more details on cardinality constraints see [20, 21]. For the integration of different user views in HERM see, e.g., [22]. Cardinality constraints are also displayed in the diagram. For example, the constraint $card$(Problem-Instance, Workflow) $= (m, m)$ states that every problem instance consists of m workflows. The cardinality constraint $card$(Mini-Batch, Problem-Instance) $= (n, n)$ states that every mini-batch consists of n problem instances. Note that a mini-batch, containing a set of problem instances, will be used in the machine learning process to ensure the generality of the learned heuristics.

We use the conceptual schema in Figure 2 to represent data on workflows, virtual machines, scheduling as well as training instances, i.e., mini-batches, used during the machine learning process. Note that the entity and relationship types in gray color capture the information on workflows, while the entity and relationship type in green color capture the information on cloud resources, i.e., virtual machines. Further, there are two relationship types Schedule and Mini-Batch in blue color that capture the information on scheduling decisions and training instances.

For example, a workflow pattern can be considered as a complex data type which may be regarded as a list of tasks and a set of edges defined by pairs of tasks. If the machine learning researchers consider the workflow pattern as the internal structure of tasks to be conceptually irrelevant for the application under development then they can define a *simple* attribute *pattern* with $tp(pattern) = DAG$. Otherwise, if the internal structure of the workflow pattern is conceptually relevant then she/he may use a *complex* attribute $pattern(\{task\}, (\{parent: task, child: task\}))$.

Model-driven approach to automatic learning heuristics. Now we present our model-driven GPHH for dynamic multi-workflow scheduling. Figure 3 shows the high-level overview of the GPHH algorithm for learning heuristics based on model-driven training instances.

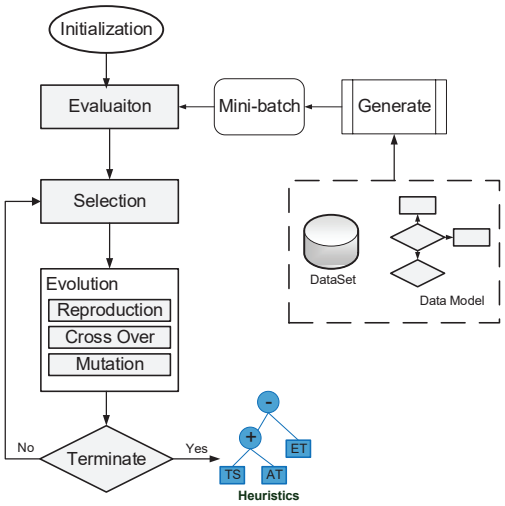


Figure 3: Model-driven machine learning for dynamic multi-workflow scheduling

Here, we represent heuristics as syntax trees, with internal nodes as arithmetic functions and leave nodes as terminals. For an example see the heuristic in Figure 3. The terminals can be either attributes or derived attributes of the object types in the ER-DMWS schema, e.g., task size *TS*. Table 1 summarizes the *terminal* set used in this case, and the *function* set including $\{+, -, \times, \div, max, min\}$ based on [23].

During the process of solving a DMWS problem instance in real time, any *GP tree* evolved by GPHH, is treated as a *priority function*. Whenever a task in a workflow becomes ready for execution, i.e., all its parent tasks have been completed, the GP tree is applied to each eligible cloud resource, e.g., a leased VM instance with the requested CPU and memory capacities to accomplish the task, to calculate their respective priorities. For this purpose, the value of each terminal node of the GP tree is determined first. Note that, for different cloud resources under consideration, the same terminal node can have varied values. With respect to any candidate cloud resource, driven by the corresponding values of all terminal nodes, we can proceed to calculate the value of each internal node of the GP tree in the bottom-up order and eventually determine the value of the root node, which will be assigned to the candidate cloud resource as its priority. After calculating the priorities of all the eligible VMs, the VM with the highest priority is then scheduled to process the task. The above process is performed repeatedly

Table 1
The terminal set

	Terminal	Definition
task-related	*TS*	The size of a task
	ET	The execution time of a task
VM-related	*CU*	The compute unit of a VM
	PRICE	The price of renting a VM for one hour
	TIQ	The total execution time of all tasks in a VM queue
	VMR	The remaining available time for a VM
	FT	The finish time of a task on a VM
	NIQ	The number of tasks in a VM queue
workflow-related	*NOC*	The number of successor tasks of a task
	NOR	The number of remaining tasks in a workflow
problem-specific	*RDL*	The remaining deadline time of a workflow

to schedule the execution of every task of each workflow, whenever a task becomes ready. Therefore, guided by a GP tree, the full schedule that solves any DMWS problem instance can be obtained. Details regarding the GPHH algorithm for DMWS can be found in [4, 16].

Apparently, different GP trees can produce varied full schedules for the same DMWS problem instance. Each full schedule also leads to varied total costs. The aim of the GPHH algorithm is to identify a GP tree that can achieve on average the smallest total cost, the objective defined in 1, for a wide variety of DMWS problem instances. For this purpose, the GPHH algorithm randomly generates a set (aka. a population) of initial GP trees (called the initial population). Each GP tree of the initial population will be evaluated using training instances and a fitness function, i.e., *TotalCost*. GP trees with good fitness values will be selected for reproduction, mutation and crossover to generate the next generation of GP trees. This process will continue until the stopping criteria are met, e.g., the maximum number of generations is reached. The evolved GP tree with the best fitness is then reported by the GPHH algorithm as its final learned heuristic for DMWS. As can be seen in Figure 3, in every generation, i.e., iteration, all the heuristics will be evaluated using the training instances. Evidently these training instances play an important role in the evolution (or heuristic learning) process.

Model-driven training instance design. The modeling technique that we have just proposed can be generalized to generate training datasets for GPHH for dynamic multi-workflow scheduling. In order to learn good heuristics that can perform well for unseen problem instances, for each generation, different training instances will be rotated, see Figure 4. However, if training instances are too different from generation to generation, then no individuals can steadily survive through generations to evolve. Therefore, we need to carefully design suitable training instances. This can be controlled by specifying a proper size for the instance pool. In the mean time, to train heuristics that perform well for unseen problem instances, we adapt the concept of 'mini-batch' [5] in the design of training instances. As can be seen in Figure 2, a mini-batch may contain one or more problem instances. We will explore settings of different

batch sizes in the section on experiments.

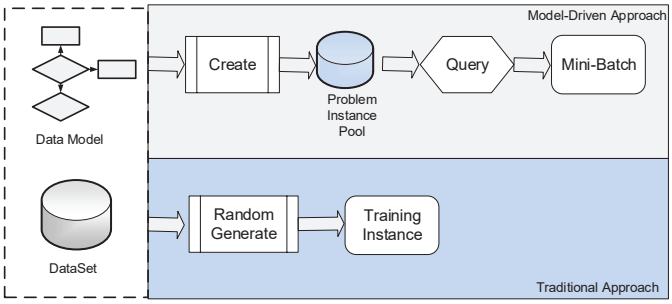
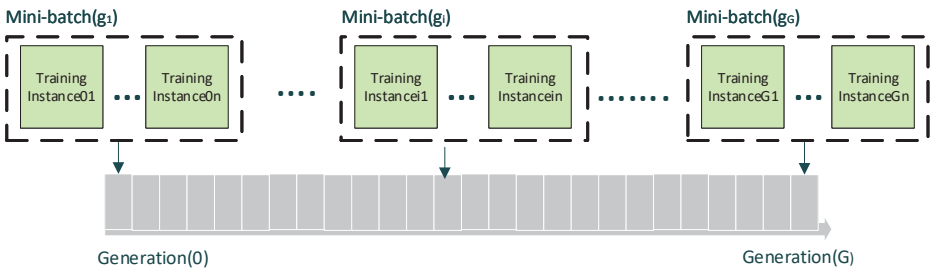
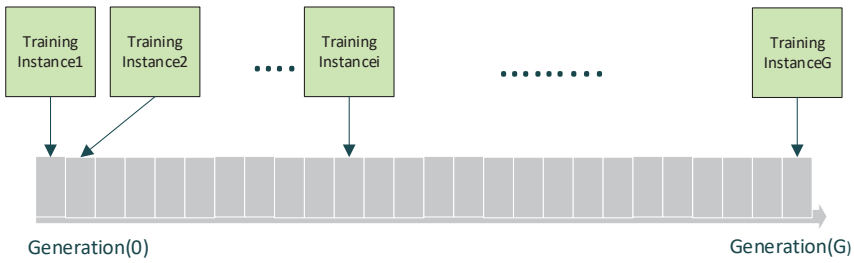


Figure 4: Generating training instances for GPHH

Figure 4 shows two possible ways to generate training instances, i.e., the model-driven and the traditional approach. The traditional approach randomly generates a problem instance consisting of n workflows following randomly selected workflow patterns and scales. Our model-driven approach creates a problem instance pool, which consists of m problem instances. For each generation, a mini-batch of training instances is created by selecting n problem instances.



(a) Rotating training instance - Model-driven approach through generations



(b) Rotating training instance - Traditional approach

Figure 5: Rotating training instance through GPHH generations

As can be seen in Figure 4, our model-driven approach first creates a problem instance pool before running GPHH. During the training process of GPHH, training instance can be obtained by querying the problem instance pool. To continue with, we provide details of each step.

Definition 1. *A dynamic multi-workflow scheduling* ***problem instance*** I_t *consists of a list of m workflows, i.e.,* $I_t = [w_{t1}, \ldots, w_{tm}]$*, where* w_{ti} *is randomly generated using available workflow patterns and sizes,* (PatternName, Size)*, and is assigned a random arrival time* ArriveTime*. A* ***problem instance pool*** $\mathbb{P}$ *consists of p problem instances, i.e.,* $\mathbb{P} = \{I_1, \ldots, I_t \ldots, I_p\}$.

As shown in Figure 2, we propose to use a mini-batch consisting of a set of problem instances for a particular generation throughout generations of the training process.

Definition 2. *A* ***mini-batch*** $B(g_i)$ *for generation* g_i *is a set of n problem instances, i.e.,*

$$B(g_i) = \{I^1(g_i), I^2(g_i), \ldots, I^k(g_i), \ldots, I^n(g_i)\}$$

where each problem instance $I^k(g_i)$ *is retrieved from the problem instance pool* $\mathbb{P}$*, i.e.,* $I^k(g_i) \in \mathbb{P}$ *for* $k = 1, \ldots, n$.

To create a problem instance pool $\mathbb{P}$, we generate a set of problem instances, using the schema defined for the DMWS problem, but each time with different combinations of patterns, sizes and arrival times. In this way, our problem instance pool contains a variety of problem instances to represent various possible problem scenarios. We can then obtain a mini-batch of n problem instances by randomly selecting n problem instances from $\mathbb{P}$.

5. A case study

To evaluate the performance of our proposed model-driven approach, we have implemented a machine learning simulator using benchmark data of workflow scheduling as in [4, 24, 25], i.e., benchmark workflow patterns with various sizes, and VM data obtained from commercial AWS data centers [26, 27]. The simulator is used to train heuristics for dynamic workflow scheduling. After the heuristics are trained using the simulator, we evaluate the heuristics using test datasets. We evaluated the performance (in terms of the total cost) of the heuristics in solving SLA-aware dynamic multi-workflow scheduling, comparing with heuristics that are trained using the training datasets generated using some existing approach. That is, two different training datasets are used, with one based on our proposed model-driven approach and the other based on the existing approach [12, 16, 28]. Two sets of heuristics are trained on each of the two training datasets. Since real-world datasets were considered, the number of workflow scheduling requests will naturally vary throughout the day and week. This may lead to increased execution time for a single workflow due to the queuing time of tasks VMs.

Datasets. We first discuss the simulation configuration, followed by the parameter settings of GPHH. The detailed setup of comparative experiments will be introduced, too. A simulated cloud environment is used to experimentally evaluate the proposed model-driven GPHH approach to DMWS. It contains several key components listed below.

VM Types: According to Amazon EC2[1], we assume that the cloud environment supports six different VM types with their respective configurations summarized in Table 2. The maximum number of VM instances of each type is unlimited.

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

Table 2
Configurations of 6 VM types 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

Workflows: Figure 6 shows four widely used workflow patterns[2], namely *CyberShake, Inspiral, Montage*, and *SIPHT*, that are employed for our experimental study. Table 3 gives detailed information of 12 heterogeneous workflows used in our experiments.

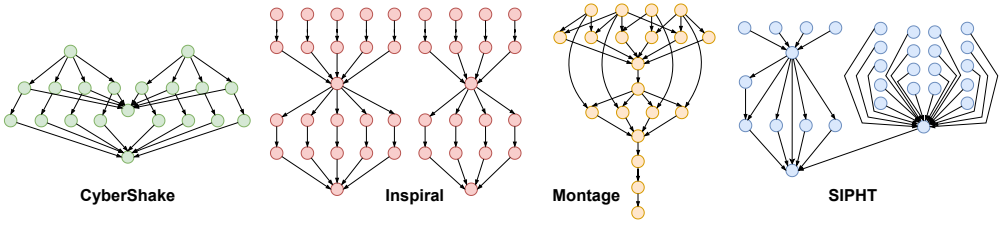


Figure 6: Four workflow patterns widely used in the state-of-the-art literature on workflow scheduling

Table 3
Information of 12 different workflow types based on Figure 6 and utilized in our experiments

Index	DAG-Size	Number of nodes	Number of edges	Average task process time (CU=1)
1	CyberShake_30	30	52	405.62 s
2	CyberShake_50	50	88	487.86 s
3	CyberShake_100	100	180	514.52 s
4	Inspiral_30	30	35	3529.10 s
5	Inspiral_50	50	60	3763.82 s
6	Inspiral_100	100	119	3363.83 s
7	Montage_25	25	45	145.76 s
8	Montage_50	50	106	162.76 s
9	Montage_100	100	233	172.69 s
10	SIPHT_30	29	33	3060.12 s
11	SIPHT_60	58	66	3219.01 s
12	SIPHT_100	97	109	2866.76 s

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

Baselines. Several baselines are utilized to provide a point of comparison for the experiments.

- **HEFT:** Heterogeneous earliest finish time [12] is a widely known heuristic algorithm for workflow scheduling. All individuals in all generations are evaluated by the same one training instance.
- **Random Generate** (no rotation): Traditional approach where the GPHH algorithm is set to use one training instance per generation and is constant across generations [15].
- **Random Generate** (rotation): Traditional approach where the GPHH algorithm is set to use one training instance per generation, and randomly generates a new training instance in each generation [7].

Parameter settings. The study utilized default parameter settings for GPHH, following the recommendations provided in [4, 29]. The population size was set to 1024, with a total of 50 iterations. One individual was designated as elite, and the tournament size was set to 7. Crossover, mutation, and reproduction rates were set at 0.80, 0.15, and 0.05, respectively. Furthermore, the initial depth of GP trees ranged from 2 to 6, and the maximum depth was constrained to 8 throughout the evolutionary process.

Evaluation results. Let **Mini-batch Select n/p** denote our proposed model-driven approach where the GPHH algorithm is set to randomly select n problem instances from a problem instance pool with p instances in each generation. Next we show the performance of the proposed model-driven GPHH approach in comparison to the baselines.

Table 4
The mean (standard deviation) of the test performance of 30 independent runs of our proposed model-driven approach using mini-batch size $n = 1$ and problem instance pool sizes $p = 10, 20, 40$ in comparison to the baselines HEFT and the traditional GPHH approaches

Test Case	HEFT	Random Generate (no rotation)	Random Generate (rotation)	Mini-batch Select 1/10 (rotation)	Mini-batch Select 1/20 (rotation)	Mini-batch Select 1/40 (rotation)
mix_all	74.03(6.47)	83.28(15.26)	66.75(4.85)	69.09(12.81)	**66.37(4.59)**	72.17(13.35)
mix_small	47.67(4.68)	40.01(7.38)	**31.65(2.19)**	33.71(6.79)	32.52(3.06)	34.98(6.81)
mix_medium	72.0(2.37)	75.05(13.53)	**60.18(4.03)**	64.5(10.45)	62.35(4.15)	66.7(11.12)
mix_large	110.25(8.85)	122.38(19.16)	**99.88(6.58)**	103.05(16.56)	99.89(6.01)	108.18(20.17)
pure_cybershake	37.05(0.63)	32.95(6.34)	**26.68(2.62)**	27.08(4.41)	26.83(3.48)	27.39(4.68)
pure_inspiral	124.55(15.77)	137.06(31.45)	119.17(7.44)	120.75(14.14)	**115.74(7.45)**	119.74(10.64)
pure_montage	36.03(0.5)	23.27(3.58)	19.28(1.54)	18.76(2.91)	**17.62(1.93)**	18.97(3.65)
pure_sipht	89.87(0.08)	103.6(26.16)	86.89(3.73)	88.28(9.61)	**85.52(4.89)**	89.99(10.97)
Average Rank	5.25	5.62	1.75	3	**1.5**	3.88

We have also conducted the Wilcoxon rank sum test with a significance level of 0.05 to make pairwise comparisons (each algorithm is compared with all the algorithms to its left in the table). The "(+)/(-)/(=)" after each entry in the table indicates that the corresponding results are significantly better/worse than or similar to the results of the compared method.

To provide insides of the performance, Figure 7 shows the convergence curves for the methods using different training instances. It can be observed that the heuristics learned using a mini-batch designed based on our model-driven approach performs better than the heuristics learned using randomly generated training instances, throughout the whole training process.

Table 5
Wilcoxon test on the approaches compared in Table 4

Test Case	HEFT	Random Generate (no rotation)	Random Generate (rotation)	Mini-batch Select 1/10 (rotation)	Mini-batch Select 1/20 (rotation)	Mini-batch Select 1/40 (rotation)
mix_all	—	(-)	(+)(+)	(+)(+)(=)	(+)(+)(=)(=)	(=)(+)(-)(=)(-)
mix_small	—	(+)	(+)(+)	(+)(+)(=)	(+)(+)(=)(=)	(+)(+)(-)(=)(=)
mix_medium	—	(=)	(+)(+)	(+)(+)(-)	(+)(+)(-)(=)	(+)(+)(-)(=)(=)
mix_large	—	(-)	(+)(+)	(+)(+)(=)	(+)(+)(=)(=)	(=)(+)(-)(=)(-)
pure_cybershake	—	(+)	(+)(+)	(+)(+)(=)	(+)(+)(=)(=)	(+)(+)(=)(=)(=)
pure_inspiral	—	(-)	(=)(+)	(=)(+)(=)	(+)(+)(=)(=)	(+)(+)(=)(=)(=)
pure_montage	—	(+)	(+)(+)	(+)(+)(+)	(+)(+)(+)(+)	(+)(+)(=)(=)(-)
pure_sipht	—	(-)	(+)(+)	(+)(+)(=)	(+)(+)(+)(=)	(+)(+)(=)(=)(=)

Table 6
The mean (standard deviation) of the test performance of 30 independent runs of our proposed model-driven approach using mini-batch sizes $n = 1, 3, 5$ and problem instance pool sizes p up to 40

Test Case	Mini-batch Select 1/20	Mini-batch Select 3/10	Mini-batch Select 3/20	Mini-batch Select 5/20	Mini-batch Select 5/40
mix_all	66.37(4.59)	62.86(4.32)(+)	64.79(6.0)(=)	**62.77(4.52)(+)**	63.87(5.2)(+)
mix_small	32.52(3.06)	30.47(2.09)(+)	31.37(3.54)(+)	**30.26(1.84)(+)**	30.7(2.01)(+)
mix_medium	62.35(4.15)	59.05(4.12)(+)	60.17(5.47)(+)	**58.99(4.07)(+)**	59.62(4.62)(+)
mix_large	99.89(6.01)	95.26(6.87)(+)	97.27(8.38)(=)	**94.72(7.0)(+)**	95.8(7.98)(+)
pure_cybershake	26.83(3.48)	26.07(1.8)(=)	26.27(2.52)(=)	**26.06(1.47)(=)**	26.26(2.12)(=)
pure_inspiral	115.74(7.45)	115.59(7.99)(=)	**113.21(7.79)(=)**	114.16(9.03)(=)	117.16(8.12)(=)
pure_montage	17.62(1.93)	17.52(1.59)(=)	18.09(2.66)(=)	**17.24(1.23)(=)**	18.14(2.18)(=)
pure_sipht	85.52(4.89)	84.26(2.95)(=)	85.06(4.42)(=)	**83.68(2.36)(+)**	84.6(2.85)(=)
+ / = / -	—	4/4/0	2/6/0	5/3/0	4/4/0
Average Rank	4.62	2.12	3.62	**1.12**	3.5

We further evaluate the performance of our proposed approach using a mini-batch with multiple problem instances. Here we consider $n = 3, 5$. Table 6 shows the evaluation results.

As we can see from Table 6, our proposed model-driven approach using a mini-batch of size larger than 1 can achieve better results than just using a mini-batch of size 1. However, even with a mini-batch of size 1, our proposed model-driven approach performs better than the existing approaches using randomly generated training instances as done in the recent literature on dynamic workflow scheduling, cf. [7], according to Table 4. This demonstrates that using the training instances generated based on the conceptual schema in Figure 2 can well support GPHH to learning better heuristics. Hence, model-driven GPHH can produce better heuristics than the traditional approach that uses randomly generated training instances.

6. Conclusions and future work

In this paper we proposed a model-driven machine learning approach to the dynamic multi-workflow scheduling (DMWS) problem. For machine learning algorithms to effectively learn optimization rules or heuristics we proposed a model-driven approach to design a mini-batch of training instances. We designed a conceptual database schema for DMWS, on which training instances can be generated in a systematic way, to cover possible problem scenarios. Simulations using benchmark and real-world data demonstrate that our proposed model-driven GPHH

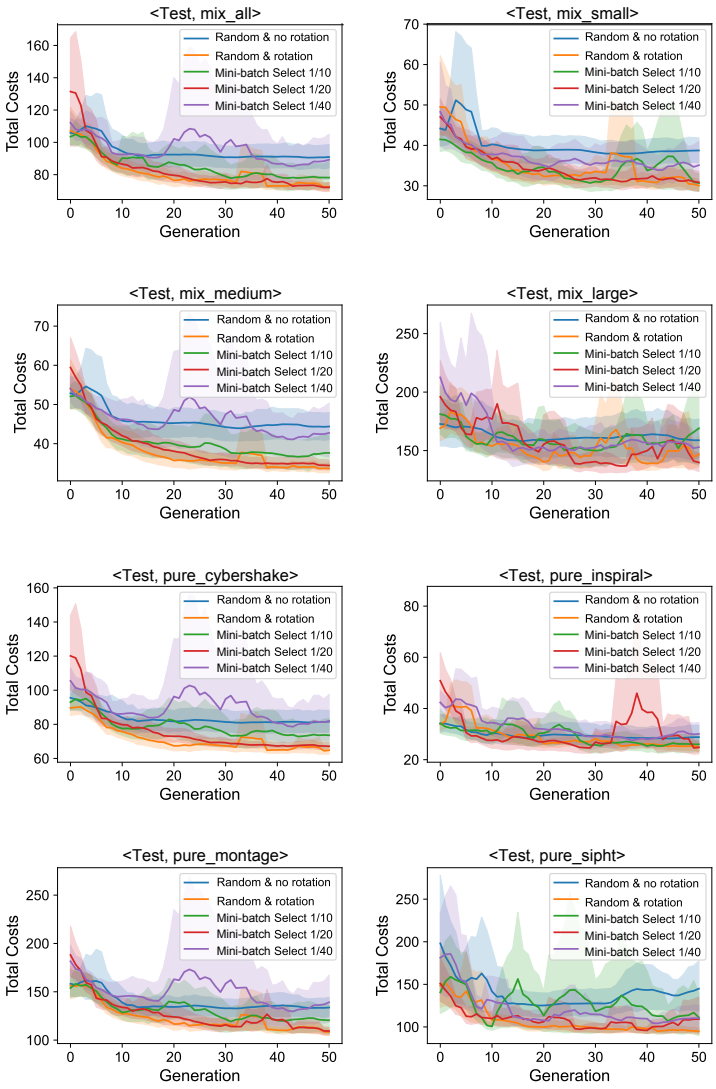


Figure 7: Convergence curves

approach outperforms existing approaches to dynamic multi-workflow scheduling problems. Our novel model-driven approach is very effective in supporting machine learning algorithms to train effective heuristics for dynamic workflow scheduling problems. Future work can be conducted to investigate the optimal choice of the mini-batch size and the problem pool size.

References

[1] A. Schrijver, et al., Combinatorial optimization: polyhedra and efficiency, volume 24, Springer, 2003.

[2] C. H. Papadimitriou, K. Steiglitz, Combinatorial optimization: algorithms and complexity, Courier Corporation, 1998.
[3] Z. Zhu, G. Zhang, M. Li, X. Liu, Evolutionary multi-objective workflow scheduling in cloud, IEEE Transactions on Parallel and Distributed Systems 27 (2015) 1344–1357.
[4] Y. Yang, G. Chen, H. Ma, M. Zhang, Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud, in: International Conference on Service-Oriented Computing (ICSOC), Springer, 2022, pp. 433–448.
[5] N. Gazagnadou, R. Gower, J. Salmon, Optimal mini-batch and step sizes for saga, in: International Conference on Machine Learning (ICML), PMLR, 2019, pp. 2142–2150.
[6] J. Wu, W. Hu, H. Xiong, J. Huan, V. Braverman, Z. Zhu, On the noisy gradient descent that generalizes as SGD, in: International Conference on Machine Learning (ICML), PMLR, 2020, pp. 10367–10376.
[7] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling, IEEE Transactions on Cybernetics 51 (2021) 1797–1811.
[8] S. Nalchigar, E. Yu, R. Ramani, A conceptual modeling framework for business analytics, in: International Conference on Conceptual Modeling (ER), Springer, 2016, pp. 35–49.
[9] S. Nalchigar, E. Yu, K. Keshavjee, Modeling machine learning requirements from three perspectives: a case report from the healthcare domain, Requirements Engineering 26 (2021) 237–254.
[10] Y. Liu, Y. Mei, M. Zhang, Z. Zhang, Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem, in: ACM Genetic and Evolutionary Computation Conference (GECCO), 2017, pp. 290–297.
[11] S. Abrishami, M. Naghibzadeh, D. H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Generation Computer Systems 29 (2013) 158–169.
[12] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (2002) 260–274.
[13] B. Tan, H. Ma, Y. Mei, M. Zhang, A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds, IEEE Trans. Cloud Comput. 10 (2022) 1500–1514.
[14] B. Tan, H. Ma, Y. Mei, A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in container-based clouds, in: IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 2681–2688.
[15] Y. Chen, T. Shi, H. Ma, G. Chen, Multi-objective location-aware service brokering in multi-cloud - a GPHH approach with transfer learning, in: International Conference on Applications of Evolutionary Computation (EvoStar), Springer, 2023, pp. 573–587.
[16] K.-R. Escott, H. Ma, G. Chen, Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud, in: International Conference on Database and Expert Systems Applications (DEXA), Springer, 2020, pp. 76–90.
[17] A. Nasiri, S. Nalchigar, E. Yu, W. Ahmed, R. Wrembel, E. Zimányi, From indicators to predictive analytics: A conceptual modelling framework, in: IFIP Working Conference on The Practice of Enterprise Modeling (PoEM)), Springer, 2017, pp. 171–186.

[18] S. Nalchigar, E. Yu, Business-driven data analytics: A conceptual modeling framework, Data & Knowledge Engineering 117 (2018) 359–372.
[19] C.-H. Youn, M. Chen, P. Dazzi, Cloud broker and cloudlet for workflow scheduling, Springer, 2017.
[20] B. Thalheim, Entity Relationship Modeling – Foundations of Database Technology, Springer, 2000.
[21] S. Hartmann, On the consistency of int-cardinality constraints, in: International Conference on Conceptual Modeling (ER), Springer, 1998, pp. 150–163.
[22] H. Ma, K. Schewe, B. Thalheim, J. Zhao, View integration and cooperation in databases, data warehouses and web information systems, J. Data Semantics (2005) 213–249.
[23] Q.-Z. Xiao, J. Zhong, L. Feng, L. Luo, J. Lv, A cooperative coevolution hyper-heuristic framework for workflow scheduling problem, IEEE Transactions on Services Computing 15 (2022) 150–163.
[24] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: IEEE International Conference on e-Science (E-Science), IEEE, 2012, pp. 1–8.
[25] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, N. Najjari, Online multi-workflow scheduling under uncertain task execution time in IaaS clouds, IEEE Transactions on Cloud Computing 9 (2019) 1180–1194.
[26] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, N. Rasouli, GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds, IEEE Transactions on Parallel and Distributed Systems 31 (2019) 1239–1254.
[27] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, P. Maechling, Scientific workflow applications on Amazon EC2, in: IEEE International Conference on e-Science (E-Science), IEEE, 2009, pp. 59–66.
[28] V. Arabnejad, K. Bubendorfer, B. Ng, Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds, Future Generation Computer Systems 100 (2019) 98–108.
[29] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Zhang, M. Zhang, Genetic programming for dynamic workflow scheduling in fog computing, IEEE Transactions on Services Computing 16 (2023) 2657–2671.