Information Sciences 678 (2024) 120975

Contents lists available at ScienceDirect

Information Sciences

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

Enhancing generalization in genetic programming hyper-heuristics through mini-batch sampling strategies for dynamic workflow scheduling

Yifan Yang [a,*], Gang Chen [a], Hui Ma [a], Sven Hartmann [b], Mengjie Zhang [a]

[a] Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand

[b] Department of Informatics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany

ARTICLE INFO

Keywords:
Dynamic workflow scheduling
Genetic programming hyper-heuristics
Generalization
Mini-batch

ABSTRACT

Genetic Programming Hyper-heuristics (GPHH) have been successfully used to evolve scheduling rules for Dynamic Workflow Scheduling (DWS) as well as other challenging combinatorial optimization problems. The method of sampling training instances has a significant impact on the generalization ability of GPHH, yet they are rarely addressed in existing research. This article aims to fill this gap by proposing a GPHH algorithm with a sampling strategy to thoroughly investigate the impact of six instance sampling strategies on algorithmic generalization, including one rotation strategy, three mini-batch strategies, and two hybrid strategies. Experiments across four scenarios with varying settings reveal that: (1) mini-batch with random sampling can outperform rotation in generalizing to unseen workflow scheduling problems under the same computational cost; (2) employing a hybrid strategy that combines rotation and mini-batch further enhances the generalization ability of GPHH; and (3) mini-batch and hybrid strategies can effectively enable heuristics trained on small-scale training instances generalizing well to large-scale unseen ones. These findings highlight the potential of mini-batch strategies in GPHH, offering improved generalization performance while maintaining diversity and suggesting promising avenues for further exploration in GPHH domains.

1. Introduction

Cloud computing is a transformative technology that enables remote access to a shared pool of high-performance computing resources and offers scalability, flexibility, and cost-efficiency [4,5]. As a result, it has been adopted by many different industries, organizations, and research institutions [3]. Scientific applications submitted to the cloud are commonly managed as *workflows*. These workflows, characterized by a set of tasks connected as a *directed acyclic graph* (DAG) [8], as shown in Fig. 1, require efficient allocation of task execution across heterogeneous *virtual machines* (VMs) hosted in the cloud. This practically important schedule decision problem is known as *workflow scheduling*. It plays a crucial role in cloud computing to minimize the cost of resource usage

* Corresponding author.
E-mail addresses: yifan.yang@tu-clausthal.de (Y. Yang), aaron.chen@tu-clausthal.de (G. Chen), hui.ma@tu-clausthal.de (H. Ma), sven.hartmann@tu-clausthal.de (S. Hartmann), mengjie.zhang@tu-clausthal.de (M. Zhang).

https://doi.org/10.1016/j.ins.2024.120975
Received 26 March 2024; Received in revised form 20 May 2024; Accepted 5 June 2024

Available online 11 June 2024

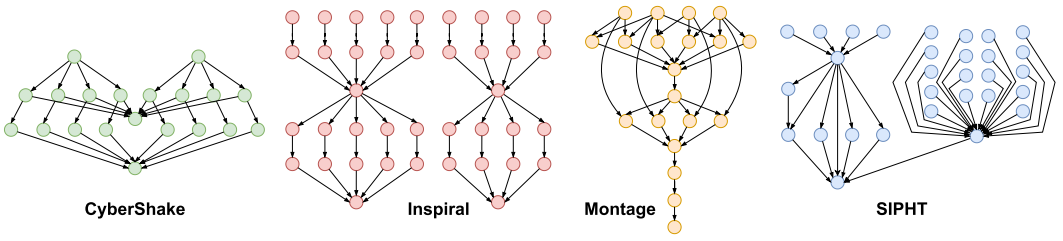

Fig. 1. Four widely used workflow patterns.

[17,35,38], meet the various quality of service requirements [1,18], and cope with the dynamic and complex nature of the cloud infrastructure [19,34,39].

This article addresses the problem of *deadline-constrained dynamic workflow scheduling in cloud computing* (DCDWSC) [17,18,40]. DCDWSC considers the dynamic arrival of a series of heterogeneous workflows with predetermined workflow patterns and deadline constraints that are scheduled to execute on multiple VMs with heterogeneous configurations. The overall goal of DCDWSC is to make intelligent decisions in terms of workflow assignment and VM provisioning to minimize the total costs associated with *VM rental fees* and *deadline violation penalties* [17,47]. To tackle such a dynamic workflow scheduling (DWS) problem, *priority-driven heuristics* are frequently employed in practice [2,23,33] due to their easy implementation and fast computation [29,44]. These heuristics iteratively schedule each workflow task to execute on the VM instance with the lowest priority value.

Genetic Programming Hyper-Heuristic (GPHH) approaches have been effectively utilized to address dynamic optimization problems such as DWS and DCDWSC problems [17,35,38,40], enabling the automatic design of priority-driven heuristics without the need for domain knowledge or human intervention. As a learning-based method, *generalization* is an important goal for GPHH [16,30], such that the evolved heuristics obtained in the *training* stage can be used to effectively solve new unseen *problem instances*.

Particularly, the *generalization* of GPHH is defined as the ability of the learned heuristic, evolved for a specific problem scenario, to achieve consistently competitive performance for multiple different problem scenarios. A *problem scenario* refers to a probability distribution over problem instances. Moreover, a *problem instance* refers to a concrete configuration of a given problem scenario/domain that needs to be solved. Problem instances are also called *training instances* or *testing instances* based on their usage stage in the algorithm.

To achieve good generalization performance, existing GPHH approaches often use the *rotation* strategy to solve optimization problems [16,28,30], which involves the use of a small batch of distinct training instances for fitness evaluation in each generation. We denote *batch size* as the number of training instances used per generation. This strategy assumes that increasing the chance of encountering different training instances under a limited number of fitness evaluations (i.e., simulation calls) can facilitate the search for heuristics with good generalization abilities [7,16].

However, there are two key issues to consider when applying the rotation strategy in DWS. First, *what batch size is appropriate for GPHH to effectively solve DWS problems?* Increasing the batch size improves the accuracy of fitness estimation at the expense of significantly increased *computational cost*. For example, in [41], a batch size of 3 required 20 hours for training over 50 generations on 10 CPUs. In practice, batch sizes are often set to very small numbers [11,28,38], such as 1 or 3, to reduce computational cost. Second, *is the method of sampling training instances that constantly uses unseen problem instances suitable for DWS problems?* Continuously rotating new problem instances to generate mini-batches of small sizes can lead to the loss of valuable heuristics during iterations [6,29]. This is due to the changes in evaluated fitness values, caused by the rotation of training instances, disrupting the stability of the heuristic learning process. Particularly for DWS, where problem instances exhibit significant differences, the rotation technique might lose its effectiveness [41].

The *mini-batch* strategy, a popular technique used in machine learning [14,42], holds promise for achieving good generalization performance within the given computational cost. This strategy assumes that reusing previously seen training instances promotes consistency in the evaluated fitness of the same GP tree across consecutive generations, thereby enabling a more reliable and stable selection of heuristics during the evolutionary process. It is achieved by sampling a mini-batch of problem instances from the training set at each generation [42]. Particularly, when the training set is extremely large, the mini-batch strategy becomes analogous to a rotation strategy.

Despite the potential benefits of mini-batch techniques, there is currently a lack of detailed empirical analysis of their impact on the generalization performance of GPHH for DWS. For example, the effects of different training set sizes, different sampling strategies, and combinations of rotation and mini-batch strategies are still largely unexplored and warrant further investigation. Furthermore, existing GPHH research lacks consensus and clear guidelines on proper training set configurations and sampling strategies for the training sets. Although various sampling strategies have been explored in existing GPHH research, such as rotating training instances [20,38] and using a small batch of instances [7,29], there is no systematic study to clarify which strategy is superior. Addressing these gaps is critical to understanding the influence of sampling strategies in GPHH and related fields. By investigating instance sampling strategies, we can improve algorithmic generalization under fixed computational cost for utilizing training sets, providing practical insights into the advancement of the GPHH domain.

This study is grounded on the hypothesis that employing different sampling strategies on the same training dataset has a significant influence on the generalization ability of GPHH. To address this hypothesis, we have formulated three key research objectives/questions: (1) Can mini-batch strategies outperform rotation in terms of generalization performance? (2) Can hybrid strategies that

combine rotation and mini-batch further enhance the generalization ability of GPHH? (3) Can the heuristics generated by mini-batch and hybrid strategies be effective for larger-scale problem instances?

The focus of this article is on the critical but understudied issue of how different training instance sampling strategies impact the generalization abilities of GPHH-generated heuristics. To this end, we use a newly proposed framework, called GPHH with an instance **S**ampling strategy (GPHHS), to examine the generalization performance of six instance sampling strategies, including one rotation [38], three mini-batch [26,29], and two novel hybrid strategies. Without designing new algorithms, we aim to provide valuable insights into existing GPHH methodologies through thorough empirical evaluations of the generalization performance of six sampling strategies. Specifically, the major contributions are listed as follows:

- We compare the performance of three mini-batch strategies with the rotation strategy given an identical budget for computational resources. The results reveal that mini-batch with a random sampling strategy outperforms rotation in terms of generalization.
- We conduct experiments with two novel hybrid strategies that combine half mini-batch and half rotation. This study represents the first systematic examination of such hybrid strategies in the literature. The results demonstrate the strong generalization capabilities of the proposed hybrid strategies, revealing the broader applicability of these hybrid approaches in GPHH.
- We evaluate the performance of mini-batch and hybrid strategies in terms of generalization to large-scale problem instances. The results show that GPHH using hybrid strategies can provide high-quality heuristics that effectively bridge the gap between problem size in training and test scenarios.
- Further analysis of population diversity shows that these instance sampling strategies have different effects on population diversity for maintaining GPHH effectiveness. Instance sampling not only profoundly affects generalization performance but is also critical for effectively producing high-quality heuristics during the evolutionary process.

The remaining sections of this article are organized as follows: Section 2 reviews related research on GPHH using rotation and mini-batch strategies. In Section 3, we formulate the DCDWSC problem and illustrate the decision-making process. Section 4 introduces the GPHH framework with instance sampling and discusses specific strategies for building mini-batch sequences. Section 5 presents experimental designs, including configuration and metrics. In Section 6, we analyze mini-batch, hybrid strategies, and scalability. Section 7 explores further analysis of population diversity. Finally, Section 9 concludes this study.

2. Related work

In this section, we review existing strategies for constructing the training set that GPHH can use to solve various decision problems. Particularly, we focus on two classes of popular and successful strategies: rotation [32,40,46] and mini-batch [24,26].

2.1. Rotation strategy

The rotation strategy is adopted by most existing GPHH studies for solving job shop scheduling [16,38], arc routing problems [26,32], and DWS problems [12,40].

The batch size plays a crucial role in achieving convergence and ensuring good performance. It is a general observation that the suitable batch sizes depend on the level of differences across all problem instances [20,26,38]. We refer to such differences as *variance*, where a large variance indicates a large difference between the problem instances. In cases with small variances, such as job shop scheduling [37,46], a batch size of 1 has been empirically shown to be sufficient [16], as the workloads of randomly created problem instances do not vary significantly. Conversely, problem scenarios with large variances typically require larger batch sizes to achieve stable convergence, as increasing the batch size helps in mitigating the impact of data/instance variance during training [21,27]. For example, arc routing problem usually uses a batch size of 5 for the evaluation in [32,26]. For DWS problems, previous studies used batch sizes of both 3 [12,41], 5 [40] or more [35].

The rotation strategy aims to improve the generalization of GPHH by maximizing the exploration of non-repetitive training instances within a limited number of generations [16,30]. However, for problems with high variance like DWS, encouraging the rotation of training instances causes the evaluated fitness evaluation values to fluctuate in adjacent generations, which may negatively affect the generalization ability of the GPHH algorithm [9].

2.2. Mini-batch strategy

In the context of GPHH, the mini-batch strategy involves a training set from which mini-batches of training instances are sampled to enable the reuse of some previously seen training instances for fitness evaluation [7,29,42].

Current GPHH research employs different methods of sampling subsets/mini-batches from the training set. For example, Nguyen et al. [29] used random sampling to construct mini-batches, each containing 5 training instances from a training set of 351 jobs, for solving resource-constrained job shop scheduling problems. Such a strategy is referred to as *mb-rand* in this article. In contrast, Liu et al. [26] tackled the uncertain capacitated arc routing problem by dividing the 90-instance training set into 18 distinct mini-batches (i.e., the batch size is $90 \div 18 = 5$). These non-overlapping mini-batches are then used sequentially during the evolutionary process. For example, the first mini-batch is used in the 1-*st* and 19-*th* generation, the second mini-batch is used in the 2-*nd* and 20-*th* generation, and so on. This type of strategy is referred to as *mb-non* in this article.

Table 1
Nomenclature.

Notations	Descriptions
τ	a problem instance
S	a problem scenario
$\mathscr{D}$	a workflow type
$\mathscr{M}$	a VM type
W	a workflow instance
V	a VM instance
$\mathcal{W}$	a set of workflow types $\mathscr{D}_i$
$\mathcal{V}$	a set of VM types $\mathscr{M}_k$
$\mathbb{W}$	a sequence of workflow instances W_i contained in τ
$\mathbb{V}$	a sequence of VM instances V_k utilized in τ
$n_{\mathcal{W}}$	the size of the workflow set $\mathcal{W}$
$n_{\mathcal{V}}$	the size of the VM set $\mathcal{V}$
m	the size of the workflow sequence $\mathbb{W}$
WL	the workload of a workflow
$\mathcal{T}_{train}$	the training set
$\mathcal{T}_{test}$	the test set
$\mathcal{B}^{(g)}$	a mini-batch of problem instances used in generation g
$\mathcal{R}$	the mini-batch sequence used in an algorithm run
n_a	the size of the training set
n_e	the size of the test set
$n_{\mathcal{B}}$	the batch size of $\mathcal{B}$
g_{max}	the maximum number of iterations

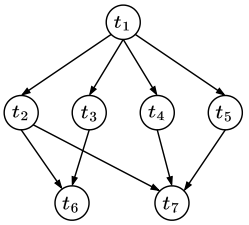

Fig. 2. Diagram of a workflow modeled as a DAG.

As seen above, different sizes of mini-batches and sampling methods (i.e., the size of the training set, whether overlapping or not) are used in the literature. To the best of our knowledge, there is no comprehensive study of the impact of mini-batch strategies on GPHH in the literature. To address this research gap, we will conduct a comprehensive series of experiments using GPHH to evolve heuristics for solving a wide range of DCDWSC problem instances, aiming to provide valuable insights and suggestions for future design of GPHH-based algorithms, especially for DWS and related problems.

3. Background

In this section, we formulate the DCDWSC problem and then illustrate the process of making scheduling decisions using a scheduling heuristic. Important notations used in our discussion are summarized in Table 1.

3.1. Problem formulation

First, a *workflow type*, formed by a workflow pattern and size, is commonly represented as a directed acyclic graph $\mathscr{D} = (T, E)$, as shown in Fig. 2. Herein, $T = \{t_i \mid i \in \{1, 2, \cdots, n_T\}\}$ is a set of *nodes* representing *tasks* in the workflow, and $E = \{e_{ij}\}$ is a set of directed *edges* where e_{ij} points from t_i to t_j. Instances of a workflow arrive dynamically at a data center to be scheduled and processed. A *workflow instance* W_i is associated with an arrival time AT_i and a deadline DL_i as well as DAG structure information such as the task size of each task TS_{ij}. In view of this, a *workflow set* is denoted as $\mathcal{W} = \{\mathscr{D}_i \mid i \in \{1, 2, \cdots, n_{\mathcal{W}}\}\}$, where $\mathscr{D}_i$ refers to a unique workflow type, and a *workflow sequence* is denoted as $\mathbb{W} = [W_i \mid i \in \{1, 2, \cdots, m\}]$, where W_i refers to a workflow instance.

A *VM type* is denoted by $\mathscr{M} = (C, P)$, where C represents the CPU (related to processing speed) and P represents the price for one hour. A *VM instance* (or *VM* in short) is of a certain type. Each VM instance, V_k is characterized by the VM identity number, rental time, its computation capacity (known as compute unit CU_k), and hourly rental fee $PRICE_k$, etc. Accordingly, a *VM set* is represented by $\mathcal{V} = \{\mathscr{M}_k \mid k \in \{1, 2, \cdots, n_{\mathcal{V}}\}\}$, where $\mathscr{M}_k$ refers to a VM type, and a *VM sequence* is represented by $\mathbb{V} = [V_k \mid k \in \{1, 2, \cdots, |\mathbb{V}|\}]$, where V_k refers to a VM instance.

A DCDWSC problem instance τ consists of a workflow sequence $\mathbb{W} = [W_1, W_2, \cdots, W_m]$ and a VM set $\mathcal{V} = \{\mathscr{M}_1, \mathscr{M}_2, \cdots, \mathscr{M}_{n_{\mathcal{V}}}\}$, and requires a heuristic to determine an optimal schedule of workflow tasks to VM instances of the given VM set. Notably, how many VM instances are rented (i.e., $|\mathbb{V}|$) to execute $\mathbb{W}$ depends on the employed scheduling heuristic h, thus even the same problem instance τ can have different $|\mathbb{V}|$ when different heuristics are used.

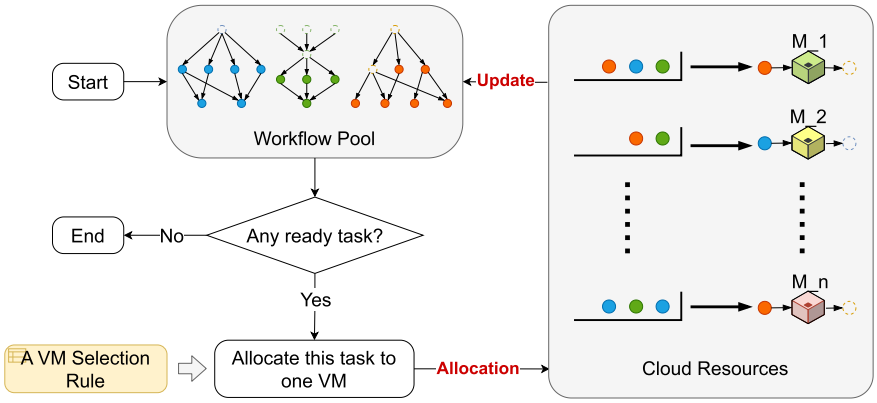

Fig. 3. Diagram of scheduling workflows to the cloud.

During the scheduling process, there are some constraints or assumptions that need to be considered:

- Each task can only be executed by one available VM, with its execution time calculated as $ET_{ij}^{k} = TS_{ij}/CU_{k}$.
- Each VM can only handle one task at a time and cannot be interrupted during execution.
- Each workflow has a deadline constraint, exceeding which will be penalized.
- Scheduling is only permitted for the task that is in the *ready* state, which means that all its predecessor tasks $pred(t_{ij})$ have been completed.
- VMs are rented hourly and automatically receive an additional hour of rental time when the current rental period is surpassed.

The *objective* of the DCDWSC problem is to minimize the *total cost* for workflow execution, consisting of *rental fees* (denoted as $RentFee$) incurred by the provision of resources and *deadline penalties* (denoted as $Penalty$) caused by violations, formulated by

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

where $\mathbb{V}$ refers to the set of leased VM instances, and $\mathbb{W}$ refers to the set of workflow instances. Particularly, two key variables introduced on the right-hand side are formulated as follows:

- $RentFee_k$ is the rental fee of k-th VM instance, defined as:

$$RentFee_k = PRICE_k \cdot \left\lceil \frac{FT_{last}^{k} - ST_{first}^{k}}{3600} \right\rceil \tag{2}$$

  where FT_{last}^{k} is the finish time of the last task on VM instance V_k, and ST_{first}^{k} is the start time of the first task on the same VM instance V_k.
- $Penalty_i$ of workflow W_i is defined as the penalty fee paid for the portion beyond its deadline DL_i:

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

  where δ is a *penalty coefficient* [43], and a larger value represents a lower tolerance for violation of the workflow deadline.

3.2. Priority-oriented heuristics for DCDWSC

Fig. 3 illustrates the process of using a priority-oriented heuristic to iteratively make VM selection decisions to solve a problem instance which includes multiple workflows and a varying set of VMs. In this article, this priority-oriented heuristic is referred to as the VM selection rule (VMSR). First, the workflow pool receives dynamically a set of workflows with various types and arrival times. Then, each ready task, defined as a workflow task for which all its predecessor tasks have been processed, will be identified to be allocated. According to a VMSR, the ready task will be allocated to a suitable VM from all VM candidates. Subsequently, each VM executes all pending tasks in its VM queue following the First-Come-First-Service principle. Whenever a pending task is processed, this information will be fed back to the workflow pool to trigger new ready tasks. The VMSR is used iteratively to make decisions until all workflows have been processed. Finally, the system outputs the total cost incurred for the workflow execution.

For the specific decision point, Table 2 gives an example of how to use a VMSR (e.g., $f(t_{ij}, V_k) = TS_{ij} + PRICE_k - ET_{ij}^{k}$) to make a decision, i.e., to select an appropriate VM for task t_{ij}. According to the terminal values in each task-VM pair, the priority values of all candidates, listed in the fifth column, can be calculated separately through the VMSR function. Finally, task t_{ij} is allocated to the VM with the minimum priority value (i.e., V_1).

Table 2
A decision situation for selecting a VM for t_{ij} by the VMSR $f(t_{ij}, V_k) = TS_{ij} + PRICE_k - ET_{ij}^k$.

Candidates	TS	PRICE	ET	$f(t_{ij}, M_k)$	Decision
V_1	2	0	1	**1**	V_1
V_2	2	7	2	7	
V_3	2	3	0.5	4.5	

Table 3
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
	$PRICE$	Price of renting a VM for one hour
	TIQ	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	Number of tasks in a VM queue
workflow-related	NOC	Number of successor tasks of a task
	NOR	Number of remaining tasks in a workflow
problem-specific	RDL	Remaining deadline time of a workflow

4. Framework design

This section begins by introducing key components of conventional GPHH, followed by the framework of GPHH with an instance sampling strategy. To facilitate understanding, we define essential concepts related to instance sampling in the proposed algorithm. Subsequently, we dive into specific strategies for building mini-batch sequences from the training set for fitness evaluation.

4.1. Components of conventional GPHH

The conventional GPHH algorithm commonly includes five key components: representation, initialization, fitness evaluation, parent selection, and evolution. Details are presented below:

Representation. Each heuristic/individual evolved by GPHH is represented as a syntax tree, also known as a GP tree, in which the leaves are *terminal* nodes and the internal nodes are *function* nodes. The set of all possible terminals is summarized in Table 3, which can be classified as task-related, VM-related, workflow-related, and problem-specific terminals. With respect to the *function* set, we consider $\{+, -, \times, \div, max, min\}$, consistent with many existing works [13,36,38].

Initialization. The individuals in the initial population are randomly generated by the *ramped-half-and-half* method [22]. In particular, half of the individuals are created by randomly adding *function* and *terminal* nodes to the GP tree until it reaches the initial depth limit. The remaining half of the individuals are created by randomly adding *function* nodes to the GP tree until it reaches the maximum tree depth.

Fitness Evaluation. Each evolved GP individual is evaluated on multiple training instances to calculate its fitness. In the g-th generation, where $g \in \{0, 1, \ldots, g_{max} - 1\}$, $n_{\mathcal{B}}$ training instances contained in $\mathcal{B}^{(g)}$ are used for evaluating the fitness of each individual $h \in P^{(g)}$, which is calculated by

$$fitness(h|\mathcal{B}^{(g)}) = \frac{1}{n_{\mathcal{B}}} \sum_{\tau \in \mathcal{B}^{(g)}} total\_cost(h|\tau), \ \forall \ h \in P^{(g)} \tag{4}$$

where $total\_cost(h|\tau)$ is the total cost incurred by using the heuristic h to solve the training instance τ.

Parent Selection. The *tournament* selection technique is employed to choose suitable parents to produce their offspring, and the tournament size is set to 7 according to [38,46]. Specifically, this means that a set of individuals is randomly picked from the population and selecting the fittest one as a parent.

Evolution. The evolution component of convectional GPHH [39,36] consists of elitism, crossover, mutation, and reproduction. 1) *Elitism* ensures that the best-performing individuals from the current generation are carried over to the next generation. In elitism, top n_e individuals in $P^{(g)}$ are directly copied to $P^{(g+1)}$. In addition, three genetic operators are employed to evolve individuals, namely reproduction, one-point crossover, and one-point mutation. 2) *Crossover* combines parts of two parent solutions to create offspring, facilitating the exchange of genetic material. Fig. 4 (a) depicts the one-point *crossover* operator. A crossover point is first chosen randomly among each of two parent individuals, and then the subtrees rooted at the chosen points of the two parents are exchanged. 3) *Mutation* introduces random changes to individuals, helping to explore new areas of the representation space and avoid local optima. Fig. 4 (b) depicts the one-point *mutation*. A mutation point is randomly chosen in the individual, and then the subtree rooted at that point is replaced with a newly generated subtree. 4) *Reproduction* involves selecting individuals based on their fitness to produce the next generation, ensuring that better-performing solutions are more likely to survive. Specifically, several individuals selected via tournament selection are directly copied to the next generation.

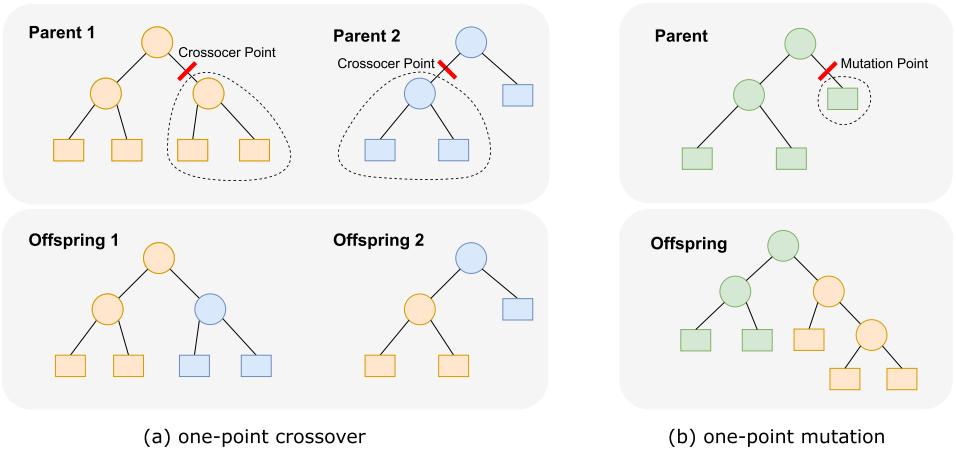

Fig. 4. Genetic operators of the GPHH algorithm.

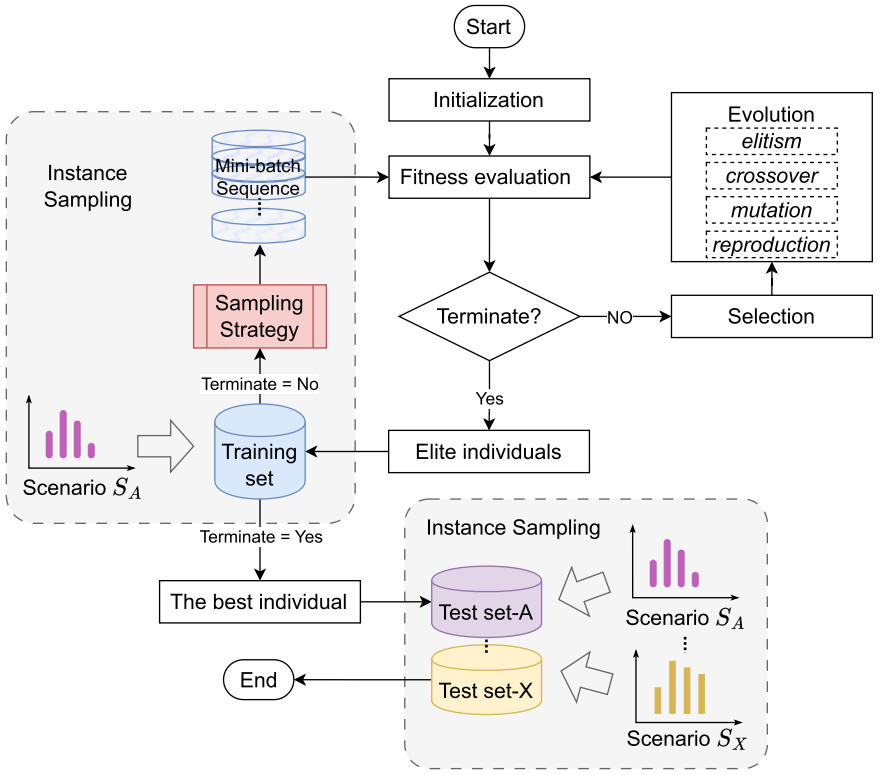

Fig. 5. Flowchart of the GPHHS algorithm.

4.2. Framework of GPHH with sampling strategies

The overall framework of GPHH with an instance **S**ampling strategy (GPHHS) is shown in Fig. 5, which consists of two parts: an *instance sampling* phase guided by an instance sampling strategy (will be elaborated in Section 4.3 and Section 4.4), and a general GPHH process encompassing representation, initialization, fitness evaluation, selection, and evolution (will be elaborated in Section 4.1).

Compared with conventional GPHH approaches, GPHHS features two key differences as follows:

(1) *Instance Sampling.* It aims to build a mini-batch sequence from one scenario (e.g., Scenario S_A) for fitness evaluation, and produce multiple test sets from different scenarios (e.g., Scenario S_A and S_X) for evaluating the best individual/heuristic. Fig. 6 shows a schematic diagram of instance sampling. To build a mini-batch sequence $\mathcal{R} = \{\mathcal{B}^{(g)}\}$, we initially create a training set $\mathcal{T}_{train}$ randomly generated from scenario S_A. Then, a mini-batch of training instances $\mathcal{B}^{(g)}$ used in generation g is sampled from $\mathcal{T}_{train}$,

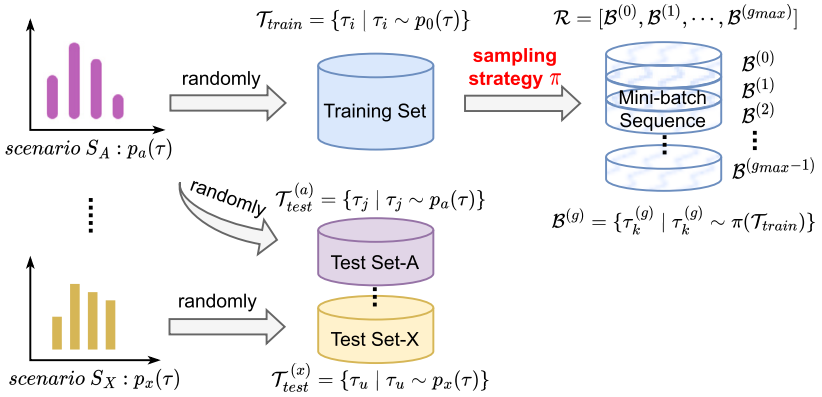

Fig. 6. Connections between different concepts within instance sampling.

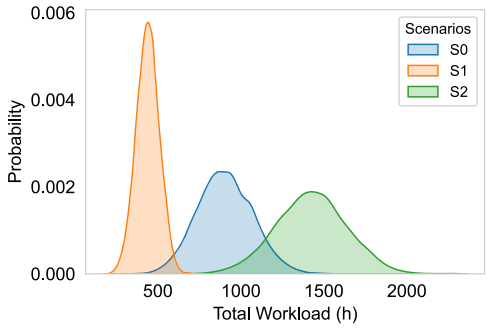

Fig. 7. Example of different problem scenarios in terms of total workload.

following an *instance sampling strategy* π introduced in Section 4.4. For test sets $\mathcal{T}_{test}^{(a)}$ and $\mathcal{T}_{test}^{(x)}$, they are randomly generated from their respective scenarios S_A and S_X. Related terms and definitions will be described in Section 4.3.

(2) *Best Individual.* The elite individuals (i.e., the top 10) obtained in the last generation will be re-evaluated on the entire training set $\mathcal{T}_{train}$, and then the best-performing individual among them will be determined as the best individual/heuristic for the training phase.

4.3. Essential concepts for instance sampling

As aforementioned in Section 3.1, $\mathcal{W} = \{\mathscr{D}_i\}$ represents a set of workflow types, and $\mathcal{V} = \{\mathscr{M}_k\}$ represents a set of VM types. In addition, $\mathbb{W} = [W_i]$ denotes a sequence of instantiated workflows, and $\mathbb{V} = [V_k]$ represents a sequence of instantiated VMs rented for executing $\mathbb{W}$ (Table 1).

Definition 1. A problem **scenario** S is a distribution over problem instances τ, denoted by

$$S : p(\tau \mid \mathcal{W}, \mathcal{V}, \theta) \tag{5}$$

with respect to a workflow set $\mathcal{W}$, a VM set $\mathcal{V}$, and additional parameter settings θ. Particularly, θ is a collection of parameters, including the workflow arrival rate λ and the deadline penalty coefficient ξ, which will be elaborated in Section 5.2 (Fig. 7).

Definition 2. A **problem instance** τ is an instantiation of a scenario that is randomly generated based on the settings in the corresponding scenario, denoted by

$$\tau = (\mathbb{W}_m, \mathcal{V}, \theta) \sim S \tag{6}$$

where $\mathbb{W}_m$ indicates a sequence of m workflow instances generated following θ (e.g., arrival rate λ). In each problem instance, $\mathbb{W}_m$ needs to be executed on the VM set $\mathcal{V}$ under the parameter setting θ (e.g., deadline penalty coefficient ξ).

Three points need explanation for Definition 2: (1) A problem instance can be used as either a training instance or a test instance. (2) In any problem instances, the allowed VM types (i.e., $|\mathcal{V}|$) are pre-determined, while the total number of allowed VM instances

(i.e., $|\mathbb{V}|$) is unlimited. (3) The workflow sequence $\mathbb{W}_m$ is formed by independently sampling m workflows one-by-one from the workflow set $\mathcal{W}$, following an identical distribution, as follows.

$$\begin{aligned}\mathbb{W}_m &= [\mathscr{D}_i + \theta \mid i \in \text{Uniform}(1, n_{\mathcal{W}}, m), \mathscr{D}_i \in \mathcal{W}] \\ &= [W_1, W_2, \cdots, W_m]\end{aligned} \tag{7}$$

where parameter m is user-defined. We set $m = 30$ in this article, following [40].

Definition 3. The **workload** of a workflow, denoted by WL, refers to the total time required for performing all tasks of the workflow one-by-one on a reference VM with a unit processing capability, calculated by

$$WL = \sum_{i:t_i \in W} TS_i \tag{8}$$

where TS_i represents the task size of task t_i of workflow W.

Definition 4. The **training set** or **test set** is a collection of problem instances randomly and independently sampled from a specific problem scenario S, represented by

$$\mathcal{T}_{train}(S) = \{\tau_1, \tau_2, \cdots, \tau_{n_a}\} \sim S \tag{9}$$

$$\mathcal{T}_{test}(S) = \{\tau_1, \tau_2, \cdots, \tau_{n_e}\} \sim S \tag{10}$$

where n_a and n_e are the size of the training set and test set, respectively. Notably, each problem instance in $\mathcal{T}_{train}$ has a unique index, serving as a reference number to build the subset for fitness evaluation in each generation $\mathcal{B}$.

Notably, the entire test set is used to assess the generalization performance of the best evolved heuristic/individual. In contrast, only a subset of the training set, sampled with an instance sampling strategy π, is utilized in any specific generation of GPHH for fitness evaluation. This subset is called a *mini-batch*, defined as follows.

Definition 5. A **mini-batch** $\mathcal{B}^{(g)}$ for generation $g \le g_{max}$ is a set of $n_{\mathcal{B}}$ problem instances obtained by an instance sampling strategy π, denoted by

$$\mathcal{B}^{(g)} = \{\tau_{b_1}^{(g)}, \tau_{b_2}^{(g)}, \cdots, \tau_{b_{n_{\mathcal{B}}}}^{(g)}\} \sim \pi(\mathcal{T}_{train}) \tag{11}$$

where $n_{\mathcal{B}}$ is the batch size, $b_{n_{\mathcal{B}}}$ is the index corresponding to any specific problem instance in $\mathcal{T}_{train}$, and π is an instance sampling strategy for sampling problem instances from the training set $\mathcal{T}_{train}$. Particularly, any $\mathcal{B}^{(g)} \subseteq \mathcal{T}_{train}$ and $n_a \le n_{\mathcal{B}} \times g_{max}$.

Definition 6. A **sequence of mini-batches** used in a GPHH evolutionary process with g_{max} generations is expressed as

$$\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}] \tag{12}$$

In the next subsection, we will investigate different strategies to sample a sequence of mini-batches $\mathcal{R}$ for GPHH, to train heuristics that can achieve high generalization performance on unseen problem instances.

4.4. Different instance sampling strategies

For the instance sampling in Fig. 5, we study six important strategies with their corresponding pseudo-code for building the mini-batch sequence $\mathcal{R}$ from the training set $\mathcal{T}_{train}$: (1) rotation (i.e., *rt*) [38], (2) mini-batch with random sampling (i.e., *mb-rand*) [14,29], (3) mini-batch without overlapping sampling (i.e., *mb-non*) [26], (4) mini-batch with overlapping sampling (i.e., *mb-lap*), (5) mini-batch combined with rotation (i.e., *mb+rt* that uses the *mb-rand* strategy first, followed by rt), and (6) rotation combined with mini-batch (i.e., *rt+mb* that uses the rotation strategy first, followed by *mb-rand*). The effectiveness of these strategies will be examined thoroughly through experiments in Section 5.

(1) *Rotation* (*rt*)

This strategy is chosen due to its common usage in current GPHH research [20,38], and its pseudo-code is presented in Algorithm 1. Given the scenario S, the batch size $n_{\mathcal{B}}$, and the number of generations g_{max}, we randomly select $n_{\mathcal{B}} \times g_{max}$ elements (i.e., problem instance τ) from the distribution S to form the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_{\mathcal{B}} \times g_{max}}$ (*line 1*). Then, in each generation g, we construct a mini-batch $\mathcal{B}^{(g)}$ by extracting $n_{\mathcal{B}}$ elements numbered from $n_{\mathcal{B}} \times g$ to $n_{\mathcal{B}} \times (g+1)$ in the training set $\mathcal{T}_{train}$ (*line 4*).

For example, given $n_{\mathcal{B}} = 3$ and $g_{max} = 4$, we can build a training set $\mathcal{T}_{train} = \{\tau_1, \tau_2, \cdots, \tau_{12}\}$. Then, the mini-batch sequence is established as $\mathcal{R} = \{\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \mathcal{B}^{(3)}\} = \{\{\tau_1, \tau_2, \tau_3\}, \{\tau_4, \tau_5, \tau_6\}, \{\tau_7, \tau_8, \tau_9\}, \{\tau_{10}, \tau_{11}, \tau_{12}\}\}$.

Algorithm 1: The rotation strategy.

Input: scenario S, batch size $n_{\mathcal{B}}$, number of generations g_{max}
Output: $\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}]$

generate the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_{\mathcal{B}} \times g_{max}} \sim S$
$g \leftarrow 0; \mathcal{R} \leftarrow [\,]$
while $g < g_{max}$ **do**
  $\mathcal{B}^{(g)} = \{\tau_i \in \mathcal{T}_{train} \mid n_{\mathcal{B}} \times g < i \leq n_{\mathcal{B}} \times (g+1)\}$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{B}^{(g)}$
  $g \leftarrow g + 1$
end

Algorithm 2: The mini-batch with random sampling strategy.

Input: scenario S, training set size n_a, batch size $n_{\mathcal{B}}$, number of generations g_{max}
Output: $\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}]$

generate the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a} \sim S$
$g \leftarrow 0; \mathcal{R} \leftarrow [\,]$
while $g < g_{max}$ **do**
  $\mathcal{B}^{(g)} = \{\tau_i \in \mathcal{T}_{train} \mid i = \text{Uniform}(1, n_a, n_B)\}$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{B}^{(g)}$
  $g \leftarrow g + 1$
end

Algorithm 3: The mini-batch without overlapping sampling strategy.

Input: scenario S, training set size n_a, batch size $n_{\mathcal{B}}$, number of generations g_{max}
Output: $\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}]$

generate the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a} \sim S$
divide $\mathcal{T}_{train}$ into $n_0 = \lfloor \frac{n_a}{n_{\mathcal{B}}} \rfloor$ parts to get $\mathcal{R}_0 = \{\mathcal{B}^{(j)}\}_{j=0}^{n_0-1}$, where $\mathcal{B}^{(j)} = \{\tau_i \in \mathcal{T}_{train} \mid n_{\mathcal{B}} \times j < i \leq n_{\mathcal{B}} \times (j+1)\}$
$g \leftarrow 0; \mathcal{R} \leftarrow [\,]$
while $g < g_{max}$ **do**
  $\mathcal{B}^{(g)} \leftarrow \mathcal{B}^{(j)} \in \mathcal{R}_0$ where $j = mod(g, n_0)$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{B}^{(g)}$
  $g \leftarrow g + 1$
end

(2) *Mini-batch with random sampling* (*mb-rand*)

This strategy is selected as it allows for the random reuse of training instances during fitness evaluation, with the potential to achieve good generalization. It has been studied in several previous research on GPHH [6,29]. Its pseudo-code is shown in Algorithm 2. In addition to S, $n_{\mathcal{B}}$ and g_{max}, the size of the training set $n_a = |\mathcal{T}_{train}|$ needs to be determined initially to form the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a}$ (*line 1*). For each generation g, a mini-batch $\mathcal{B}^{(g)}$ is created by randomly selecting $n_{\mathcal{B}}$ distinct problem instances from $\mathcal{T}_{train}$ (*line 4*).

For example, with the size of the training set size n_a being 6, we can construct $\mathcal{T}_{train} = \{\tau_1, \tau_2, \cdots, \tau_6\}$. Given $n_{\mathcal{B}} = 3$ and $g_{max} = 4$, a mini-batch sequence can be determined as $\mathcal{R} = \{\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \mathcal{B}^{(3)}\} = \{\{\tau_1, \tau_2, \tau_5\}, \{\tau_4, \tau_5, \tau_3\}, \{\tau_3, \tau_1, \tau_2\}, \{\tau_6, \tau_2, \tau_4\}\}$.

(3) *Mini-batch without overlapping sampling* (*mb-non*)

This strategy is chosen as it is a variant of the mini-batch technique proposed in [26], which does not overlap training instances between adjacent generations to increase instance diversity. The pseudo-code is shown in Algorithm 3. The training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a}$ is partitioned into $n_0 = \lfloor \frac{n_a}{n_{\mathcal{B}}} \rfloor$ non-overlapping partitions (*line 2*). Each partition, denoted by $\mathcal{A}^{(j)}$ with index j $(0 \leq j \leq n_0 - 1)$, contains $n_{\mathcal{B}}$ problem instances from $\tau_{n_{\mathcal{B}} \times j}$ to $\tau_{n_{\mathcal{B}} \times (j+1)}$ in $\mathcal{T}_{train}$. All partitions jointly form the sequence $\mathcal{R}$. In each generation g, a specific $\mathcal{A}^{(j)}$ is used as the mini-batch $\mathcal{B}^{(g)}$ where $j = mod(g, n_0)$ (*line 5*).

Given $n_a = 7, n_{\mathcal{B}} = 3$ and $g_{max} = 4$, we have $\mathcal{T}_{train} = \{\tau_1, \tau_2, \cdots, \tau_7\}$, and then partition it into $n_0 = \lfloor \frac{7}{3} \rfloor = 2$ parts to form $\mathcal{R}_0 = \{\{\tau_1, \tau_2, \tau_3\}, \{\tau_4, \tau_5, \tau_6\}\}$. Consequently, the mini-batch sequence is $\mathcal{R} = \{\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \mathcal{B}^{(3)}\} = \{\mathcal{R}_0[0], \mathcal{R}_0[1], \mathcal{R}_0[0], \mathcal{R}_0[1]\} = \{\{\tau_1, \tau_2, \tau_3\}, \{\tau_4, \tau_5, \tau_6\}, \{\tau_1, \tau_2, \tau_3\}, \{\tau_4, \tau_5, \tau_6\}\}$.

(4) *Mini-batch with overlapping sampling* (*mb-lap*)

This strategy is proposed to mitigate training instance variability in fitness evaluation as it ensures a more stable and reliable evaluation of algorithm performance across different generations. It is realized by changing one problem instance per generation. It is a new variant of the mini-batch strategy and presents its pseudo-code in Algorithm 4. Given the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a}$, a moving window of size $n_{\mathcal{B}}$ traverses $\mathcal{T}_{train}$ with $step = 1$ in each generation g, capturing a fragment as the mini-batch $\mathcal{B}^{(g)}$ (*line 4*). The captured fragment consists of problem instances with index $i \in \{mod(g + k, n_a) + 1\}_{k=0}^{n_{\mathcal{B}}-1}$ in $\mathcal{T}_{train}$.

For example, we can produce $\mathcal{T}_{train} = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ with $n_a = 4$. Given $n_{\mathcal{B}} = 3$ and $g_{max} = 4$, the mini-batch sequence is $\mathcal{R} = \{\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \mathcal{B}^{(3)}\} = \{\{\tau_1, \tau_2, \tau_3\}, \{\tau_2, \tau_3, \tau_4\}, \{\tau_3, \tau_4, \tau_1\}, \{\tau_4, \tau_1, \tau_2\}\}$.

Algorithm 4: The mini-batch with overlapping sampling strategy.

Input: scenario S, training set size n_a, batch size n_B, number of generations g_{max}
Output: $\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}]$
generate the training set $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{n_a} \sim S$
$g \leftarrow 0; \mathcal{R} \leftarrow [\,]$
while $g < g_{max}$ **do**
  $\mathcal{B}^{(g)} = \{\tau_i \in \mathcal{T}_{train} \mid i \in \{mod(g+k, n_a)+1\}_{k=0}^{n_B-1}\}$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{B}^{(g)}$
  $g \leftarrow g+1$
end

Algorithm 5: The mini-batch combined with rotation strategy.

Input: scenario S, training set size of the *mb-rand* strategy $n_a^{(mb)}$, batch size n_B, number of generations g_{max}
Output: $\mathcal{R} = [\mathcal{B}^{(0)}, \mathcal{B}^{(1)}, \cdots, \mathcal{B}^{(g_{max}-1)}]$
training set size of the *rt* strategy $n_a^{(rt)} = n_B \times (g_{max} - \lceil \frac{g_{max}}{2} \rceil)$
generate the first half training set $\mathcal{T}_{train}^{f} = \{\tau_i\}_{i=1}^{n_a^{(mb)}} \sim S$
generate the last half training set $\mathcal{T}_{train}^{l} = \{\tau_j\}_{j=1}^{n_a^{(rt)}} \sim S$
$g \leftarrow 0; \mathcal{R} \leftarrow [\,]$
while $g < g_{max}$ **do**
  if $g \leq \lceil \frac{g_{max}}{2} \rceil$ **then**
    $\mathcal{B}^{(g)} = \{\tau_i \in \mathcal{T}_{train}^{f} \mid i = \text{Uniform}(1, n_a^{(mb)}, n_B)\}$
  else
    $g' = g - \lceil \frac{g_{max}}{2} \rceil$
    $\mathcal{B}^{(g)} = \{\tau_j \in \mathcal{T}_{train}^{l} \mid n_B \times g' < j \leq n_B \times (g'+1)\}$
  end
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{B}^{(g)}$
  $g \leftarrow g+1$
end

(5) *Mini-batch combined with rotation* (*mb+rt*)

We propose a new hybrid strategy to explore the potential effects of combining two different sampling approaches, aiming to leverage the strengths of both methods and potentially achieve superior generalization in GPHH. It sequentially combines the *mb-rand* strategy and the *rt* strategy. Specifically, given a total of g_{max} generations, from generation 0 to generation $\lceil \frac{g_{max}}{2} \rceil - 1$, the *mb-rand* strategy is adopted. Subsequently, from generation $\lceil \frac{g_{max}}{2} \rceil$ onward, the *rt* strategy is used. The pseudo-code of this hybrid strategy, denoted as *mb+rt*, is shown in Algorithm 5. Different from the strategies introduced above, the *mb+rt* strategy uses two separate training sets, i.e., $\mathcal{T}_{train}^{f}$ for the *mb+rt* strategy and $\mathcal{T}_{train}^{l}$ for the *rt* strategy, with $\mathcal{T}_{train}^{f} \cap \mathcal{T}_{train}^{l} = \emptyset$.

(6) *Rotation combined with mini-batch* (*rt+mb*)

This is another hybrid strategy denoted as *rt+mb* proposed by us, aiming to use different orders to combine the two methods as a comparison of the *mb+rt* strategy. From generation 0 to generation $\lceil \frac{g_{max}}{2} \rceil - 1$, the *rt* strategy is used. From generation $\lceil \frac{g_{max}}{2} \rceil$ onward, the *mb-rand* strategy is adopted. We omit the pseudo-code for this hybrid strategy since it is similar to Algorithm 5 except that *rt* and *mb-rand* strategies are used with a different order.

5. Experimental design

This section introduces the detailed settings of the experiment, including the simulation environment configuration in Section 5.1, specific parameter settings in Section 5.2, and employed performance metrics in Section 5.3.

5.1. Simulation configuration

A simulated cloud environment[1] is used to measure the evolved heuristics (i.e., VMSRs). The simulation environment includes several key components listed below.

VM Set: According to Amazon EC2,[2] the cloud environment supports 6 different VM types (i.e., $n_{\mathcal{V}} = 6$) with their respective configurations summarized in Table 4. The maximum number of VM instances of each type is unlimited.

Workflow Set: Consistent with many previous studies [8,17,38], this article uses four workflow patterns (shown in Fig. 1) with three sizes (a.k.a., *small, medium, large*), forming a total of $4 \times 3 = 12$ different workflow types (i.e., $n_{\mathcal{W}} = 12$). Table 5 presents the information of each workflow type.

Problem Instance: Referring to the common setting [36,40], each training or test instance involves the processing of 30 workflows that are randomly sampled from the workflow set $\mathcal{W}$ of a specific problem scenario, i.e., $m = |\mathbb{W}| = 30$.

[1] https://github.com/YifanYang1995/Simulator-Dynamic-Workflow-Scheduling-in-Cloud-Computing.git.
[2] https://aws.amazon.com/ec2/pricing/on-demand/.

Table 4
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

Table 5
Information of different workflow types.

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

Table 6
Components of the workflow set in each problem scenario. (* corresponds to the index of the workflow type in Table 5.)

Scenario	Scenario Name	$^*\mathcal{W}$	$n_{\mathcal{W}}$	$n_{\mathcal{V}}$
S_0	*mix_all*	1-12	12	6
S_1	*mix_small*	1-4	4	6
S_2	*mix_medium*	5-8	4	6
S_3	*mix_large*	9-12	4	6

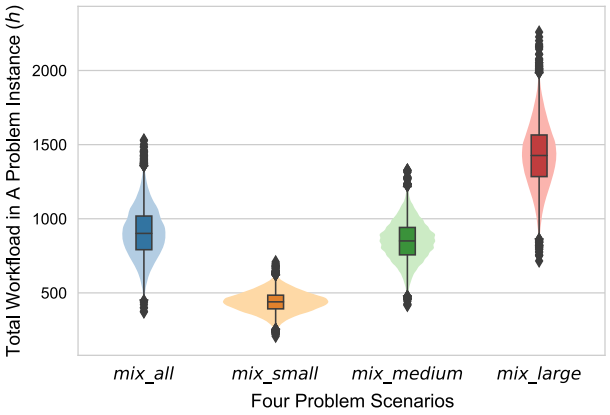

Fig. 8. Violin plot of the total workload of a problem instance under four problem scenarios over 10,000 problem instances.

Problem Scenario: Table 6 outlines four distinct problem scenarios that are frequently used in the workflow scheduling domain [17,38], covering mixed workflow types. For the VM set $\mathcal{V}$, all scenarios use the same six VM types listed in Table 4. In scenario *mix_all* (S_0), all 12 workflow types in Table 5 are considered; In scenario *mix_small* (S_1), 4 workflow types with a number of tasks around 30 are considered; In scenario *mix_medium* (S_2),4 workflow types are considered, each with approximately 50 tasks; In scenario *mix_large* (S_3), 4 workflow types with a number of tasks around 100 are considered. Fig. 8 illustrates the distinctions between four problem scenarios in terms of the total workload of each problem instance. In each scenario, we randomly generate 10,000 problem instances, and then calculate the total workload of each problem instance. We can observe that the total workload distribution of problem instances in all problem scenarios approximately follows a Gaussian distribution. This is consistent with the practical findings that the job execution time on high-performance computing systems often obeys a Gaussian distribution [2,36].

Table 7
Configurations of $\langle name, n_a \rangle$ for six strategies (e.g., $\langle mb\text{-}lap, 10 \rangle$ stands for GPHH that uses the *mini-batch with overlapping* sampling strategy, and the size of the training set is 10. $\langle rt+mb, 20 \rangle$ represents GPHH using the hybrid strategy, where the first 50 generations use $\langle rt, \infty \rangle$ and the subsequent 50 generations follow $\langle mb\text{-}rand, 20 \rangle$.)

name	n_a		Computational cost
rt	∞		$n_B \times g_{max} \times ps = 310,272$
mb-rand	{10, 20, 30, 40}		
mb-non	{15, 30, 45, 60}		
mb-lap	{5, 10, 15, 20}		
name	$n_a(g \leq 50)$	$n_a(g > 50)$	
mb+rt	{10, 20, 30, 40}	∞	
rt+mb	∞	{10, 20, 30, 40}	

Training Set: Only problem scenario *mix_all* (S_0) is employed to build the training set in our experiments. The choice of S_0 is because its workflow set contains all workflow types that occur in the other scenarios S_1, S_2, and S_3. Additionally, S_0 offers a comprehensive representation of the diverse workflow types encountered in real-world applications. The varying settings of the training set size n_a for six strategies are listed in Table 7. The corresponding experiments are labeled as $\langle name, n_a \rangle$, referring to the strategy name and the training set size. All experiments have a similar computational cost measured in terms of the total number of independent simulations performed, i.e., $3 \times 101 \times 1024 = 310272$.

Test Set: All scenarios listed in Table 6 are employed to construct multiple test sets for assessing the generalization performance of the trained heuristics. The size of each test set is set to $n_e = 50$, i.e., 50 test instances are randomly sampled from the corresponding problem scenario.

5.2. Other settings

Request Generation: Our experiments follow parameter settings widely adopted in previous studies [17,39] in the DWS field. Particularly, workflows arrive at a cloud data center over time following a Poisson distribution with $\lambda = 0.01$ [25]. The penalty coefficient in Eq. (3) is set to the unit price of the most expensive VM in Table 4, i.e., $\delta = \$2.304/h$. In addition, the deadline DL_i of each workflow is set to 1/4 of its makespan on unit-speed VMs, following [17,39].

GPHHS: The parameters of GPHHS in this study adhere to the default values recommended in [22,38]. The population size is $ps = 1024$, the number of generations is $g_{max} = 101$, the number of elites is 10, and the tournament size is 7. The crossover, mutation, and reproduction rates are 0.80, 0.15, and 0.05, respectively. In addition, the initial depth of GP trees ranges from 2 to 6, and the maximum depth is limited to 8 during the evolutionary process [46].

5.3. Performance metrics

Three important metrics are used to analyze all experiment results. Below we briefly introduce and justify the use of each metric.

•*Test performance*

Equation (13) is used to measure how well the learned heuristic performs on a test set that is different from the training set. The performance of a specific heuristic h on a specific test set $\mathcal{T}_{test}^{(i)}$ is evaluated by the average total cost achieved by h across 50 test instances, formulated by

$$Perf(h|\mathcal{T}_{test}^{(i)}) = \frac{1}{50} \sum_{j=1}^{50} total\_cost(h|\tau_j \in \mathcal{T}_{test}^{(i)}) \tag{13}$$

where $\mathcal{T}_{test}^{(i)}$ is the test set of scenario S_i. The lower the value of $Perf(h)$, the better the test performance of h. Moreover, a Wilcoxon signed-rank test with a significance level of 0.05 is performed to compare the test performance across these strategies.

•*Generalization ability*

The generalization ability, denoted as $General$, is a scalar that measures the average test performance of a heuristic h trained in scenario S_0 across multiple scenarios (i.e., S_0, S_1, S_2 and S_3). The Friedman test in terms of test performance $Perf(h|\cdot)$ is used to calculate the generalization of GPHH, calculated by

$$General(h) = \frac{1}{4} \sum_{i=1}^{4} rank^{(i)} \tag{14}$$

where $rank^{(i)}$ is the ranking of $Perf(h|\mathcal{T}_{test}^{(i)})$ among all competing heuristics. Specifically, a smaller value of $General(h)$ indicates better generalization performance.

•*Population diversity*

This metric is used to quantify the diversity of GP evolved heuristics within a single GP population. Existing studies often use the *phenotypic characterization* (PC) to measure the population diversity [15,37,45].

Table 8
Example of obtaining the PC vector of two heuristics. (The PC vectors of two heuristics upon 3 decision situations are $PC(h_1) = (2,2,2)^T$ and $PC(h_2) = (3,2,1)^T$, respectively.)

Candidates	reference rule h_{ref}	h_1	$PC(h_1)$	h_2	$PC(h_2)$
V_1	2	①	2	2	3
V_2	1	2		3	
V_3	3	3		①	
V_1	1	2	2	2	2
V_2	2	①		①	
V_3	4	4		3	
V_4	3	3		4	
V_1	1	2	2	①	1
V_2	3	3		2	
V_3	2	①		3	

A VM selection decision situation refers to a specific time or event during the scheduling process where a decision needs to be made to select a VM instance from candidate VMs. Following [37], we randomly select 40 VM selection decision situations to obtain the PC vector of each heuristic, denoted by $PC(h_i) = (x_1^{(i)}, x_2^{(i)}, \cdots, x_{40}^{(i)})^T$. In particular, $x^{(i)}$ is the corresponding rank of the highest-ranked VM instance by h_i on the reference rule h_{ref}. Table 8 gives an example of how to obtain the PC vectors of heuristic h_1 and h_2 in 3 decision situations.

Given the PC vectors of any two heuristics h_i and h_j, we can calculate their *phenotypic distance* by the Hamming distance [10]:

$$d_{(i,j)} = \sum_{k=1}^{40} \varepsilon_k, \quad \text{where } \varepsilon_k = \begin{cases} 0, & \text{if } x_k^{(i)} = x_k^{(j)} \\ 1, & \text{otherwise} \end{cases} \tag{15}$$

For example, the phenotypic distance of h_1 and h_2 in Table 8 is 2. After obtaining the PC vectors of all heuristics/individuals in a population $\mathcal{P}$, we define *population diversity* as the averaged phenotypic distance between any two individuals, calculated by

$$Diversity(\mathcal{P}) = \frac{2}{|\mathcal{P}|(|\mathcal{P}|+1)} \sum_{h_i, h_j \in \mathcal{P}} d_{(i,j)}, \quad \text{where } i < j \tag{16}$$

Particularly, a large value indicates high population diversity.

6. Results and analysis

In this section, we empirically investigate three components: mini-batch strategies, hybrid strategies, and scalability on large-scale problem instances. For each component, we present the research question, experiments, results, and observations.

6.1. Mini-batch strategies

•*Question*: Can mini-batch strategies outperform rotation in terms of generalization performance?

•*Experiments*: We compare the generalization performance of heuristics generated by GPHH using one rotation and three mini-batch strategies defined in Section 4.4: *rt* [38], *mb-rand* [14,29], *mb-non* [26], and *mb-lap*. We analyze the performance of different mini-batch strategies across varying training set sizes n_a, as shown in Table 7. In addition, the batch size is $n_{\mathcal{B}} = 3$ according to the common practice [41]. Particularly, $\langle rt, \infty \rangle$ is treated as a baseline with the same computational cost as the mini-batch-related experiments.

•*Results*: Table 9 reports the test performance (i.e., mean and standard deviation of the total cost) of *rt, mb-rand, mb-non* and *mb-lap* on four problem scenarios across 30 independent runs. Notably, all these experiments consume the same number of simulations during one run, approximately requiring 18 hours on 15-CPU.

It can be observed that $\langle$*mb-rand*, 20$\rangle$ is the best configuration regarding the generalization ability, representing that 3 training instances are randomly sampled from a training set of size 20 in each generation for fitness evaluation. Although $\langle$*mb-rand*, 20$\rangle$ is not significantly better than $\langle rt, \infty \rangle$ in terms of test performance across four problem scenarios, it shows improvements regarding the test performance with a much smaller training set than that of $\langle rt, \infty \rangle$. This is valuable for solving practical optimization problems where data collection (i.e., training instances) is expensive or difficult.

Furthermore, in mini-batch-related experiments, we can observe that the size of the training set is not linearly related to the generalization performance of the GPHH algorithm. How the sampling is conducted appears to be more crucial, such as random sampling, with or without overlapping.

Upon comparing experiments of three mini-batch strategies, it is evident that mini-batch with random sampling outperforms the other two variants. Particularly, the *mb-lap* strategy achieves the worst test performance and generalization. For *mb-lap*, there exists significant overlap between adjacent subsets, i.e., $|\mathcal{B}^{(i)} \cap \mathcal{B}^{(i+1)}| = n_{\mathcal{B}} - 1$. For *mb-non*, it prohibits overlapping between adjacent subsets, i.e., $\mathcal{B}^{(i)} \cap \mathcal{B}^{(i+1)} = \emptyset$. As for the *mb-rand* strategy, it can be viewed as an intermediary strategy since $0 \le |\mathcal{B}^{(i)} \cap \mathcal{B}^{(i+1)}| \le n_{\mathcal{B}}$.

Table 9
The mean (standard deviation) test performance of all experiments on four problem scenarios with $m = 30$ workflows. ((+), (≈) or (−) indicates that the matching result is significantly better, equivalent or worse to $\langle rt, \infty \rangle$.)

$\langle name, n_a \rangle$	*mix_all*	*mix_small*	*mix_medium*	*mix_large*	*General*	+/≈/−
$\langle rt, \infty \rangle$	66.39(3.73)	37.06(3.69)	62.42(4.09)	97.32(5.61)	3.5	—
$\langle mb\text{-}rand, 10 \rangle$	67.96(3.56)(≈)	38.77(3.82)(−)	64.13(3.81)(≈)	99.91(6.73)(≈)	11.75	0/3/1
$\langle mb\text{-}rand, 20 \rangle$	**65.93(4.57)**(≈)	**36.45(4.16)**(≈)	**61.60(4.79)**(≈)	96.95(4.91)(≈)	**1.25**	0/4/0
$\langle mb\text{-}rand, 30 \rangle$	67.32(4.00)(≈)	37.56(3.45)(≈)	63.00(4.30)(≈)	98.11(4.92)(≈)	5.5	0/4/0
$\langle mb\text{-}rand, 40 \rangle$	66.37(3.76)(≈)	37.28(4.11)(≈)	62.21(4.00)(≈)	97.30(4.24)(≈)	3	0/4/0
$\langle mb\text{-}non, 15 \rangle$	67.88(4.21)(≈)	37.82(4.45)(≈)	63.11(4.83)(≈)	99.36(5.24)(≈)	8.75	0/4/0
$\langle mb\text{-}non, 30 \rangle$	66.90(3.84)(≈)	37.87(3.31)(≈)	62.78(3.98)(≈)	98.27(8.60)(≈)	6	0/4/0
$\langle mb\text{-}non, 45 \rangle$	67.50(3.60)(−)	37.97(3.84)(−)	63.18(3.81)(≈)	98.69(5.34)(≈)	8.75	0/2/2
$\langle mb\text{-}non, 60 \rangle$	66.21(4.23)(≈)	37.59(4.26)(≈)	62.23(5.92)(≈)	**96.75(4.93)**(≈)	3	0/4/0
$\langle mb\text{-}lap, 5 \rangle$	71.26(4.39)(−)	39.80(3.69)(−)	66.21(4.85)(−)	104.34(5.36)(−)	13	0/0/4
$\langle mb\text{-}lap, 10 \rangle$	67.44(4.07)(≈)	37.91(4.67)(≈)	63.06(4.51)(≈)	99.63(6.73)(≈)	8.25	0/4/0
$\langle mb\text{-}lap, 15 \rangle$	67.70(4.56)(≈)	38.60(4.84)(≈)	63.53(4.54)(≈)	100.10(10.01)(≈)	11	0/4/0
$\langle mb\text{-}lap, 20 \rangle$	67.58(3.77)(≈)	37.27(4.20)(≈)	63.33(4.60)(≈)	98.46(4.57)(≈)	7.25	0/4/0

Table 10
The mean (standard deviation) test performance of all experiments on four problem scenarios with $m = 30$ workflows. ((+), (≈) or (−) indicates that the matching result is significantly better, equivalent or worse to $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$.)

$\langle name, n_a \rangle$	*mix_all*	*mix_small*	*mix_medium*	*mix_large*	*General*	+/≈/−
$\langle rt, \infty \rangle$	66.39(3.73)	37.06(3.69)	62.42(4.09)	97.32(5.61)	8.75	—
$\langle mb\text{-}rand, 20 \rangle$	65.93(4.57)(≈)	36.45(4.16)(≈)	61.60(4.79)(≈)	96.95(4.91)(≈)	5.5	0/4/0
$\langle mb+rt, 10 \rangle$	66.19(3.86)(≈)(≈)	37.49(4.23)(≈)(−)	62.17(4.47)(≈)(≈)	96.58(3.77)(≈)(≈)	8.5	0/7/1
$\langle mb+rt, 20 \rangle$	**64.31(4.77)**(+)(+)	**35.77(4.42)**(≈)(+)	**60.31(4.99)**(+)(+)	**94.22(5.74)**(+)(+)	**1**	**7/1/0**
$\langle mb+rt, 30 \rangle$	65.73(3.77)(≈)(≈)	37.44(4.40)(≈)(≈)	61.56(4.13)(≈)(≈)	95.95(4.79)(≈)(≈)	5.25	0/8/0
$\langle mb+rt, 40 \rangle$	64.99(3.62)(+)(≈)	36.01(4.03)(≈)(≈)	60.48(4.02)(+)(≈)	95.86(4.23)(+)(≈)	2.25	3/5/0
$\langle rt+mb, 10 \rangle$	66.29(4.21)(≈)(≈)	36.96(3.69)(≈)(≈)	61.96(4.47)(≈)(≈)	96.66(5.29)(≈)(≈)	8.75	0/8/0
$\langle rt+mb, 20 \rangle$	65.76(4.25)(≈)(≈)	37.07(4.09)(≈)(≈)	61.58(4.59)(≈)(≈)	95.88(4.72)(≈)(≈)	5.25	0/8/0
$\langle rt+mb, 30 \rangle$	66.25(4.07)(≈)(≈)	37.13(3.95)(≈)(≈)	61.90(4.53)(≈)(≈)	96.56(4.47)(≈)(≈)	6.75	0/8/0
$\langle rt+mb, 40 \rangle$	65.34(4.49)(≈)(≈)	36.51(4.27)(≈)(≈)	61.31(4.86)(≈)(≈)	95.37(5.02)(≈)(≈)	3	0/8/0

This suggests a more balanced way to explore and exploit training instances, which might be the reason why it is better than the *mb-non* and *mb-lap* strategies.

•*Observations:* Under the same computational cost, the mini-batch with random sampling strategy can yield better generalization of GPHH compared with the rotation strategy, resulting in moderate improvement in the test performance.

6.2. Hybrid strategies

•*Question*: Can hybrid strategies further enhance the generalization ability of GPHH?

•*Experiments*: We assess the performance of two hybrid strategies, *mb+rt* and *rt+mb*, under the same computation cost as Section 6.1. Particularly, *mb+rt* indicates half *mb-rand* followed by half *rt*, and *rt+mb* indicates half *rt* followed by half *mb-rand*. As shown in Table 7, sensitivity analysis regarding the training set size is performed for both hybrid strategies, where $n_a^{(mb)}$ denote the training set size of the *mb-rand* strategy.

•*Results*: Table 10 records the mean (standard deviation) of test performance of two hybrid strategies *mb+rt* and *rt+mb* on four problem scenarios across 30 independent runs. Particularly, $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$ are treated as baselines. Among them, the configuration with the best generalization ability is $\langle mb+rt, 20 \rangle$.

In terms of test performance, $\langle mb+rt, 20 \rangle$ and $\langle mb+rt, 40 \rangle$ significantly outperform $\langle rt, \infty \rangle$ across three problem scenarios, i.e., *mix_all*, *mix_medium* and *mix_large*. Moreover, $\langle mb+rt, 20 \rangle$ is significantly better than $\langle mb\text{-}rand, 20 \rangle$ in all four problem scenarios. We further calculate the averaged improvement ratio of test performance across four scenarios of $\langle mb\text{-}rand, 20 \rangle$ and $\langle mb+rt, 20 \rangle$ relative to $\langle rt, \infty \rangle$. The values are 1.01% and 3.29% respectively, indicating that the hybrid algorithm further improves the generalization ability of GPHH using compared with the mini-batch with random sampling.

Overall, employing a hybrid strategy – *whether starting with rotation or mini-batch strategies* – to sample the training set yields superior test performance compared with only using rotation. Notably, in several cases, it yields significantly improved results. These highlight the advantages of using the hybrid strategy, including enhanced generalization capability and reduced sensitivity to parameter settings.

•*Observations*: The combined use of the rotation and mini-batch strategies in GPHH holds a high potential to enhance the test performance (i.e., generalization performance) of GPHH, compared with using the rotation strategy alone.

6.3. Scalability on large-scale problem instances

•*Question*: Can the heuristics generated by mini-batch and hybrid strategies be effective for larger-scale problem instances?

Table 11
The mean (standard deviation) test performance of experiments on four problem scenarios with $m = 50$ workflows. ((+), (≈) or (−) indicates that the matching result is significantly better, equivalent or worse to $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$.)

$\langle name, n_a \rangle$	*mix_all*	*mix_small*	*mix_medium*	*mix_large*	*General*	+/ ≈ /−
$\langle rt, \infty \rangle$	118.27(12.55)	66.93(9.39)	112.14(12.74)	184.63(37.13)	6	—
$\langle mb\text{-}rand, 20 \rangle$	114.40(13.27)(≈)	64.23(9.46)(≈)	108.21(12.96)(≈)	173.01(17.63)(≈)	3	0/4/0
$\langle mb+rt, 20 \rangle$	111.13(12.05)(+)(+)	**62.71(9.31)**(+)(+)	105.30(11.91)(+)(+)	**169.03(15.52)**(+)(+)	**1.5**	**8/0/0**
$\langle mb+rt, 40 \rangle$	**110.98(10.67)**(+)(≈)	62.87(8.81)(+)(≈)	**104.24(10.81)**(+)(≈)	169.30(14.65)(+)(≈)	**1.5**	4/4/0
$\langle rt+mb, 20 \rangle$	116.31(12.62)(≈)(≈)	65.88(8.85)(≈)(≈)	109.94(12.26)(≈)(≈)	175.77(17.84)(≈)(≈)	5	0/8/0
$\langle rt+mb, 40 \rangle$	114.74(13.05)(≈)(≈)	64.85(9.59)(≈)(≈)	108.34(12.76)(≈)(≈)	174.14(18.30)(≈)(≈)	4	0/8/0

Table 12
The mean (standard deviation) test performance of experiments on four problem scenarios with $m = 70$ workflows. ((+), (≈) or (−) indicates that the matching result is significantly better, equivalent or worse to $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$.)

$\langle name, n_a \rangle$	*mix_all*	*mix_small*	*mix_medium*	*mix_large*	*General*	+/ ≈ /−
$\langle rt, \infty \rangle$	153.04(18.62)	87.6(11.74)	149.39(20.29)	262.00(214.64)	6	—
$\langle mb\text{-}rand, 20 \rangle$	146.99(13.30)(≈)	84.16(11.72)(≈)	142.99(14.71)(≈)	218.06(14.64)(≈)	3	0/4/0
$\langle mb+rt, 20 \rangle$	**143.75(12.69)**(+)(+)	**82.97(12.19)**(≈)(≈)	**140.03(13.64)**(+)(≈)	**213.15(15.16)**(+)(+)	**1**	**5/3/0**
$\langle mb+rt, 40 \rangle$	144.81(11.63)(+)(≈)	83.89(11.65)(≈)(≈)	140.42(13.87)(+)(≈)	216.03(12.45)(≈)(≈)	2	2/6/0
$\langle rt+mb, 20 \rangle$	148.34(11.56)(≈)(≈)	86.21(10.23)(≈)(≈)	144.26(12.53)(≈)(≈)	218.56(14.31)(≈)(≈)	4.25	0/8/0
$\langle rt+mb, 40 \rangle$	149.45(17.95)(≈)(≈)	85.82(11.85)(≈)(≈)	144.77(17.42)(≈)(≈)	222.52(31.92)(≈)(≈)	4.75	0/8/0

•*Experiments*: We assess the test performance and generalization ability of heuristics trained on small-scale instances (i.e., $m = 30$ workflows) directly on larger-scale scenarios. Specifically, the number of workflows contained in a problem instance m is increased to 50 and 70 workflows for testing in four problem scenarios, i.e., *mix_all*, *mix_small*, *mix_medium* and *mix_large*.

•*Results*: Table 11 ($m = 50$) and Table 12 ($m = 70$) show the mean and standard deviation of test performance of six configurations in four scenarios. Two configurations serve as baselines: $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$. In addition, four well-performing configurations of the *hybrid* strategy are considered: $\langle mb+rt, 20 \rangle$, $\langle mb+rt, 40 \rangle$, $\langle rt+mb, 20 \rangle$, and $\langle rt+mb, 40 \rangle$.

Table 11 shows that all configurations of the *hybrid* strategy perform better than $\langle rt, \infty \rangle$ in terms of generalization. Particularly, $\langle mb+rt, 20 \rangle$ and $\langle mb+rt, 40 \rangle$ demonstrate significantly superior test performance compared with $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$ across all scenarios. Furthermore, the test performances of $\langle mb+rt, 20 \rangle$ are 6.89% and 2.55% better than those of $\langle rt, \infty \rangle$ and $\langle mb\text{-}rand, 20 \rangle$, respectively.

Table 12 shows that $\langle mb+rt, 20 \rangle$ is significantly better than $\langle rt, \infty \rangle$ in three scenarios (i.e., *mix_all*, *mix_medium* and *mix_large*), while $\langle mb+rt, 40 \rangle$ significantly outperforms it in two (i.e., *mix_all* and *mix_medium*). In addition, while not significantly superior to $\langle mb\text{-}rand, 20 \rangle$ in all scenarios, they still demonstrate commendable test performance based on the mean and standard deviation values. For example, in scenario *mix_large*, $\langle mb+rt, 20 \rangle$ demonstrates both a lower total cost and a smaller standard deviation than that of $\langle mb\text{-}rand, 20 \rangle$. This highlights the effectiveness of using the *hybrid* strategy in GPHH approaches.

Under the same computational cost, the *mb+rt* strategy enables the evolved heuristics to have significantly better test performance in most high-workload problem scenarios than using the rotation strategy. Although not significantly outperforming *mb-rand* in many scenarios, *mb+rt* does improve test performance by about 2.73% in terms of the mean values.

•*Observations*: Compared with the rotation strategy, heuristics trained using the mini-batch and hybrid strategies in small-scale scenarios exhibit superior generalization performance on large-scale scenarios.

7. Further analysis

This section analyzes the impact of fitness evaluation criteria in Eq. (4) resulting from different sampling strategies on population diversity. Population diversity is crucial for maintaining the effectiveness of GPHH. This is because diverse populations facilitate the exploration of a wider heuristic space, leading to the discovery of more robust and adaptable heuristics.

According to the metric defined in Eq. (16), we perform the population diversity analysis on strategies $\langle rt, \infty \rangle$, $\langle mb\text{-}rand, 20 \rangle$, $\langle mb+rt, 20 \rangle$ and $\langle rt+mb, 40 \rangle$. Fig. 9 presents the population diversity curves across the evolutionary process. Higher values indicate greater phenotypic distances between heuristics, implying higher population diversity. It can be observed that all instance sampling strategies maintain high population diversity in the early stage of evolution, with varying degrees of decline in the later stage. This pattern suggests that initial exploration efforts yield diverse heuristics, but as the evolutionary process progresses, convergence towards promising heuristics leads to a reduction in diversity.

Compared to $\langle rt, \infty \rangle$, we can see that curves of $\langle mb\text{-}rand, 20 \rangle$, $\langle mb+rt, 20 \rangle$ and $\langle rt+mb, 40 \rangle$ strongly reduce the population diversity after 50 generations. Compared with $\langle mb\text{-}rand, 20 \rangle$, two hybrid strategies slightly enhance population diversity in the last 20 generations. Consequently, the population diversity must be traded in a reasonable range to contribute to the effectiveness of the algorithm [9]. Furthermore, instance sampling not only has a profound impact on the generalization performance, but also plays a vital role in effectively producing high-quality heuristics/rules during the evolutionary process. We suggest using the hybrid strategy that provides a middle ground by maintaining some diversity while improving generalization compared to rotation and mini-batch strategies.

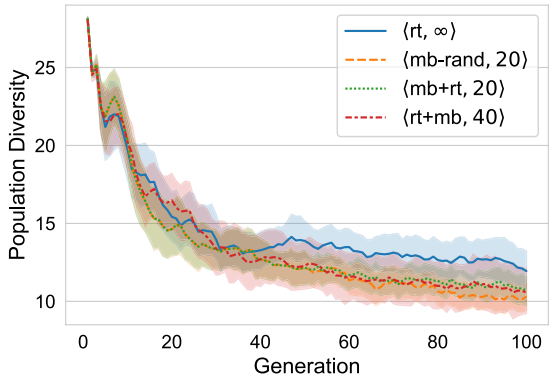

Fig. 9. Curves of the population diversity in $\langle rt, \infty \rangle$, $\langle mb\text{-}rand, 20 \rangle$, $\langle mb + rt, 20 \rangle$ and $\langle rt + mb, 40 \rangle$.

8. Discussion

In this study, we investigated the impact of various instance sampling strategies on the generalization ability of GPHH for DWS problems. We compared the performance of one rotation strategy, three mini-batch strategies, and two novel hybrid strategies across different problem scales and scenarios.

Our research findings provide the following recommendations for practitioners when applying instance sampling strategies in GPHH: (1) Using the mini-batch with random sampling strategy improves the generalization ability of GPHH; (2) Leverage Hybrid Strategies: Combining rotation and mini-batch strategies enhances GPHH's generalization performance significantly compared to using rotation alone; and (3) Heuristics trained on small-scale scenarios using hybrid strategies perform consistently better than using rotation or mini-batch alone, especially regarding their performance on large-scale scenarios.

Our study contributes to the existing literature by offering a systematic examination of instance sampling strategies in the context of GPHH. While previous studies have used instance sampling strategies in an ad hoc way, our research provides comprehensive experiments to verify their effectiveness. By rigorously evaluating the performance of different strategies across various problem scales and scenarios, we contribute to advancing the understanding of GPHH methodologies.

Despite the strengths of our study, several limitations should be acknowledged. While our proposed strategies have demonstrated significant improvements in most scenarios, we acknowledge that achieving consistent results across all scenarios can be challenging. In future work, we can leverage machine learning techniques, such as transfer learning, to further enhance the generalization ability of GPHH. For example, we can fine-tune the GPHH model trained on an existing scenario to improve its performance on any new scenarios. Furthermore, this study aims to improve generalization under the given computational cost. While the trade-off between generalization and computational costs is also an important consideration, it falls beyond the scope of this study. We will consider exploring this trade-off in our future research endeavors. For example, we can dynamically adjust the number of generations and training instances used for fitness evaluation during the evolutionary process based on changes in the generalization behavior of the evolved heuristics.

Our research has significant implications for the design and implementation of GPHH algorithms. By highlighting the importance of instance sampling strategies in enhancing generalization performance, we provide valuable insights for practitioners seeking to optimize heuristic search algorithms for complex scheduling problems.

9. Conclusions

This study thoroughly investigated the impact of various instance sampling strategies, including one rotation strategy, three mini-batch strategies and two novel hybrid strategies, on the generalization performance of GPHH. We formally defined essential concepts such as scenario, problem instance, and workload. We further proposed the corresponding training framework named GPHH with an instance sampling strategy (GPHSS). Guided by it, we experimentally examined six instance sampling strategies on three problem scales in four scenarios.

This study marks the first systematic examination of the hybrid strategy that combines mini-batch with rotation in the literature. Empirical experiments demonstrated using mini-batch strategies can enhance the generalization ability of GPHH for DWS. In addition, the hybrid strategies can further provide high-quality heuristics in tackling complex DWS problems, which enhance generalization ability and reduce sensitivity to parameter settings. Furthermore, the mini-batch and hybrid strategies demonstrate strong abilities in effectively scaling to large-scale unseen problem instances.

Future research can explore adaptive mini-batch designs and investigate their applicability to other problem domains. In addition, the idea derived from the use of prioritized replay in the deep Q-network variant [31] can also be employed to enhance the performance of GPHH methods. By prioritizing certain training instances, GPHH can be used to evolve heuristics that can effectively solve large-scale and complex problems.

CRediT authorship contribution statement

Yifan Yang: Writing – original draft, Methodology, Investigation. **Gang Chen:** Writing – review & editing, Supervision, Methodology. **Hui Ma:** Writing – review & editing, Supervision, Methodology. **Sven Hartmann:** Writing – review & editing, Supervision. **Mengjie Zhang:** Writing – review & editing, Supervision, Conceptualization.

Declaration of competing interest

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

Data availability

Data will be made available on request.

References

[1] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.
[2] S.G. Ahmad, C.S. Liew, M.M. Rafique, E.U. Munir, S.U. Khan, Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems, in: IEEE Fourth International Conference on Big Data and Cloud Computing, IEEE, 2014, pp. 129–136.
[3] V. Arabnejad, K. Bubendorfer, B. Ng, Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds, Future Gener. Comput. Syst. 100 (2019) 98–108.
[4] S.A. Bello, L.O. Oyedele, O.O. Akinade, M. Bilal, J.M.D. Delgado, L.A. Akanbi, A.O. Ajayi, H.A. Owolabi, Cloud computing in construction industry: use cases, benefits and challenges, Autom. Constr. 122 (2021) 103441.
[5] A. Benlian, W.J. Kettinger, A. Sunyaev, T.J. Winkler, G. Editors, The transformative value of cloud computing: a decoupling, platformization, and recombination theoretical framework, J. Manag. Inf. Syst. 35 (3) (2018) 719–739.
[6] Y. Bi, B. Xue, M. Zhang, Instance selection-based surrogate-assisted genetic programming for feature learning in image classification, IEEE Trans. Cybern. (2021).
[7] Y. Bi, B. Xue, M. Zhang, Using a small number of training instances in genetic programming for face image classification, Inf. Sci. 593 (2022) 488–504.
[8] W. Chen, E. Deelman, Workflowsim: a toolkit for simulating scientific workflows in distributed environments, in: IEEE International Conference on E-Science, IEEE, 2012, pp. 1–8.
[9] S.Y. Chong, P. Tino, X. Yao, Relationship between generalization and diversity in coevolutionary learning, IEEE Trans. Comput. Intell. AI Games 1 (3) (2009) 214–232.
[10] W.S. Du, Subtraction and division operations on intuitionistic fuzzy sets derived from the Hamming distance, Inf. Sci. 571 (2021) 206–224.
[11] K.-R. Escott, H. Ma, G. Chen, Transfer learning assisted gphh for dynamic multi-workflow scheduling in cloud computing, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2022, pp. 440–451.
[12] K.-R. Escott, H. Ma, G. Chen, Cooperative coevolutionary genetic programming hyper-heuristic for budget constrained dynamic multi-workflow scheduling in cloud computing, in: Evolutionary Computation in Combinatorial Optimization: European Conference, EvoCOP 2023, Springer, 2023, pp. 146–161.
[13] D. Farinati, I. Bakurov, L. Vanneschi, A study of dynamic populations in geometric semantic genetic programming, Inf. Sci. 648 (2023) 119513.
[14] N. Gazagnadou, R. Gower, J. Salmon, Optimal mini-batch and step sizes for saga, in: International Conference on Machine Learning, in: PMLR, 2019, pp. 2142–2150.
[15] T. Hildebrandt, J. Branke, On using surrogates with genetic programming, Evol. Comput. 23 (3) (2015) 343–367.
[16] T. Hildebrandt, J. Heger, B. Scholz-Reiter, Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2010, pp. 257–264.
[17] V. Huang, C. Wang, H. Ma, G. Chen, K. Christopher, Cost-aware dynamic multi-workflow scheduling in cloud data center using evolutionary reinforcement learning, in: International Conference on Service-Oriented Computing, Springer, 2022, pp. 449–464.
[18] M. Hussain, L.-F. Wei, A. Rehman, F. Abbas, A. Hussain, M. Ali, Deadline-constrained energy-aware workflow scheduling in geographically distributed cloud data centers, Future Gener. Comput. Syst. 132 (2022) 211–222.
[19] G. Ismayilov, H.R. Topcuoglu, Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing, Future Gener. Comput. Syst. 102 (2020) 307–322.
[20] A. Jajoo, Y.C. Hu, X. Lin, N. Deng, A case for task sampling based learning for cluster job scheduling, IEEE Trans. Cloud Comput. (2022).
[21] N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P.T.P. Tang, On large-batch training for deep learning: generalization gap and sharp minima, arXiv preprint, arXiv:1609.04836, 2016.
[22] W.B. Langdon, R. Poli, Foundations of Genetic Programming, Springer Science & Business Media, 2013.
[23] Z. Li, V. Chang, H. Hu, H. Hu, C. Li, J. Ge, Real-time and dynamic fault-tolerant scheduling for scientific workflows in clouds, Inf. Sci. 568 (2021) 13–39.
[24] B.M. Lima, N. Sachetti, A. Berndt, C. Meinhardt, J.T. Carvalho, Adaptive batch size cgp: improving accuracy and runtime for cgp logic optimization flow, in: Genetic Programming: European Conference, EuroGP 2023, Springer, 2023, pp. 149–164.
[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 Trans. Cloud Comput. 9 (3) (2019) 1180–1194.
[26] Y. Liu, Y. Mei, M. Zhang, Z. Zhang, Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 290–297.
[27] D. Masters, C. Luschi, Revisiting small batch training for deep neural networks, arXiv preprint, arXiv:1804.07612, 2018.
[28] Y. Mei, S. Nguyen, B. Xue, M. Zhang, An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming, IEEE Trans. Emerg. Top. Comput. Intell. 1 (5) (2017) 339–353.
[29] S. Nguyen, D. Thiruvady, M. Zhang, D. Alahakoon, Automated design of multipass heuristics for resource-constrained job scheduling with self-competitive genetic programming, IEEE Trans. Cybern. 52 (9) (2021) 8603–8616.
[30] S. Nguyen, M. Zhang, K.C. Tan, Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules, IEEE Trans. Cybern. 47 (9) (2016) 2951–2965.
[31] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, arXiv preprint, arXiv:1511.05952, 2015.
[32] S. Wang, Y. Mei, M. Zhang, A multi-objective genetic programming algorithm with α dominance and archive for uncertain capacitated arc routing problem, IEEE Trans. Evol. Comput. (2022).

[33] X. Xia, H. Qiu, X. Xu, Y. Zhang, Multi-objective workflow scheduling based on genetic algorithm in cloud environment, Inf. Sci. 606 (2022) 38–59.
[34] J.-P. Xiao, X.-M. Hu, W.-N. Chen, Dynamic cloud workflow scheduling with a heuristic-based encoding genetic algorithm, in: International Conference on Neural Information Processing, Springer, 2020, pp. 38–49.
[35] Q.-z. Xiao, J. Zhong, L. Feng, L. Luo, J. Lv, A cooperative coevolution hyper-heuristic framework for workflow scheduling problem, IEEE Trans. Serv. Comput. 15 (1) (2019) 150–163.
[36] Q.-z. Xiao, J. Zhong, L. Feng, L. Luo, J. Lv, A cooperative coevolution hyper-heuristic framework for workflow scheduling problem, IEEE Trans. Serv. Comput. 15 (1) (2022) 150–163.
[37] M. Xu, Y. Mei, F. Zhang, M. Zhang, A semantic genetic programming approach to evolving heuristics for multi-objective dynamic scheduling, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2023, pp. 403–415.
[38] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Zhang, M. Zhang, Genetic programming for dynamic workflow scheduling in fog computing, IEEE Trans. Serv. Comput. (2023).
[39] Y. Yang, G. Chen, H. Ma, S. Hartmann, M. Zhang, Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing, IEEE Trans. Evol. Comput. (2024) 1–15, https://doi.org/10.1109/TEVC.2024.3392968.
[40] 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, Springer, 2022, pp. 433–448.
[41] Y. Yang, H. Ma, G. Chen, S. Hartmann, A model-driven machine learning approach to dynamic multi-workflow scheduling, in: Proceedings of the International Conference on Conceptual Modeling, 2023.
[42] Z. Yang, C. Wang, Z. Zhang, J. Li, Mini-batch algorithms with online step size, Knowl.-Based Syst. 165 (2019) 228–240.
[43] C.-H. Youn, M. Chen, P. Dazzi, Cloud Broker and Cloudlet for Workflow Scheduling, Springer, Singapore, 2017.
[44] C. Zhang, W. Song, Z. Cao, J. Zhang, P.S. Tan, X. Chi, Learning to dispatch for job shop scheduling via deep reinforcement learning, Adv. Neural Inf. Process. Syst. 33 (2020) 1621–1632.
[45] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Phenotype based surrogate-assisted multi-objective genetic programming with brood recombination for dynamic flexible job shop scheduling, in: IEEE Symposium Series on Computational Intelligence, IEEE, 2022, pp. 1218–1225.
[46] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, K.C. Tan, Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling, IEEE Trans. Evol. Comput. 25 (4) (2021) 651–665.
[47] L. Zhang, L. Zhou, A. Salah, Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments, Inf. Sci. 531 (2020) 31–46.