Dual-Tree Genetic Programming with Adaptive Mutation for Dynamic Workflow Scheduling in Cloud Computing

Yifan Yang, Gang Chen, *Senior Member, IEEE,* Hui Ma, *Senior Member, IEEE,* Sven Hartmann, *Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Dynamic workflow scheduling (DWS) is a challenging and important optimization problem in cloud computing, aiming to execute multiple heterogeneous workflows on dynamically leased virtual machine resources to satisfy user-defined Quality of Service requirements. For the popular deadline-constrained DWS in cloud problem, a virtual machine selection rule (VMSR) and a task selection rule (TSR) need to be designed simultaneously to minimize the rental fee and deadline violation penalty. For this purpose, Dual-Tree Genetic Programming (DTGP) has been previously developed to automatically evolve effective VMSRs and TSRs. However, existing DTGP approaches assume that VMSR and TSR, as well as terminals used by VMSRs and TSRs are equally important and evolve both VMSRs and TSRs in a black box manner, i.e., without using any knowledge about different impacts of trees and terminals. Several recent studies clearly indicate that different trees or terminals have varied performance impacts, making it critical to develop adaptive mutation mechanisms for effective DTGP. Driven by this motivation, this paper proposes two new levels of adaptive mutation mechanisms, contributing to the development of a new DTGP algorithm, which features the use of three new probability vectors for adaptive tree selection of VMSR and TSR at the first level and adaptive terminal selection at the second level while mutating any existing dual-tree individuals. Extensive experimental results demonstrate that the proposed two adaptive mechanisms can improve the effectiveness of DTGP compared to four baseline algorithms.**

***Index Terms*—dynamic workflow scheduling (DWS), deadline constraint, dual-tree genetic programming (DTGP), hyper-heuristics, adaptive mutation.**

I. Introduction

CLOUD computing is increasingly popular in numerous industries for *workflow execution* [1]. For example, MetService in New Zealand employs numerous cloud resources to run its workflows to forecast daily weather. Compared with using limited local resources, executing workflows on cloud is scalable, allowing a large volume of workflows to be executed quickly and efficiently. Furthermore, it has the flexibility to rent different types of VM instances for any duration of time to meet the different requirements of both CPU-intensive and I/O-intensive workflows.

Y. Yang, G. Chen, H. Ma, and M. Zhang are with the School of Engineering and Computer Science & Centre for Data Science and Artificial Intelligence, Victoria University of Wellington, Wellington 6012, New Zealand. E-mail: {yifan.yang, aaron.chen, hui.ma, mengjie.zhang}@ecs.vuw.ac.nz.

S. Hartmann is with the Department of Informatics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany. E-mail: sven.hartmann@tu-clausthal.de.

Workflow scheduling is a vital and challenging optimization problem in cloud computing, aiming to allocate workflow tasks to VM instances to achieve some important objectives, such as minimizing *cost* and minimizing *makespan*. Meanwhile, a service level agreement (SLA) is often used to specify various *Quality of Service* (QoS) requirements to be fulfilled, such as workflow *deadline* [2].

This paper studies the workflow scheduling problem in cloud computing from the perspective of a *broker* [3]. In brief, the broker receives workflow scheduling requests from customers and executes workflow tasks in selected cloud computing resources, i.e., *virtual machine* (VM) instances, rented from cloud providers such as Amazon EC2 [4]. As shown in Fig. 1, the broker rents VM instances from Amazon EC2 for workflow execution and then provides the execution results to its customers. The broker will compensate its customers in the case of any deadline violations. This problem is named the *deadline-constrained dynamic workflow scheduling in cloud* (DCDWSC) [5], with the goal of jointly minimizing *VM rental fees* and *deadline violation penalties*. It involves two decision-making points, *VM Selection* and *Task Selection*. One is to select an appropriate VM instance for executing each workflow task, and the other is to select a pending task to be executed on an idle VM instance.

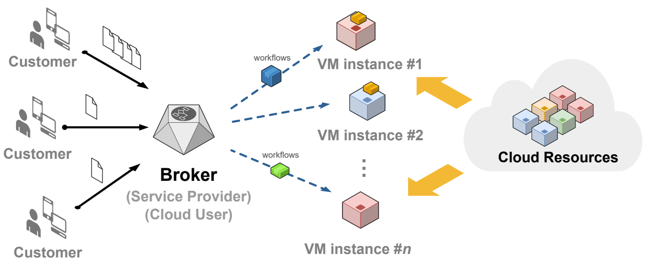


Fig. 1. Workflow scheduling from the perspective of a broker.

Several categories of approaches have been proposed to solve dynamic workflow scheduling problems. Firstly, *list-based heuristics* are mainly designed for executing a single workflow based on domain experts' knowledge [6]–[8]. However, this approach primarily centers on the allocation of tasks across a predetermined set of VM instances [7] and cannot support dynamic resource provisioning required by the DCDWSC problem.

Secondly, *scheduling heuristics/rules*, such as First-Come-First-Service (FCFS) [9], MAXMIN and MINMIN [10], have been proposed to solve dynamic workflow scheduling (DWS) problems, as they are easy to implement and efficient in making just-in-time scheduling decisions. However, these scheduling heuristics are typically manually designed, not effective or not appropriate for addressing constrained problems encountered in practice [9]. Furthermore, manually designing such scheduling heuristics require extensive human labor and domain expertise.

Thirdly, *hyper-heuristic* approaches can be used to automatically design heuristics/rules and have gained increasing attention for job shop scheduling (JSS) [11], resource allocation [12], and arc routing problems [13]. Among them, *genetic programming hyper-heuristic* (GPHH) has shown its effectiveness to evolve scheduling rules for workflow scheduling problems. Existing works [14]–[16] only evolve *VM selection rules* (VMSRs) for selecting an appropriate VM instance to execute each workflow task, without paying full attention to optimize the selection of pending task for an idle VM instance. That is, each VM instance simply follows the First Come First Serve (FCFS) principle to execute all of its pending tasks. To overcome this limitation, the GPHH approaches that can directly generate two scheduling rules have attracted widespread attention in recent years. Among them, *Cooperative Coevolution GP* (CCGP) [14], [15] and *Dual-Tree GP* (DTGP) [5], [17] are two representative approaches. Notably, experimental results in [5], [17] indicated that DTGP, which explicitly considers the inter-dependencies between the two evolved rules, outperforms CCGP for solving the DCDWSC problem. Therefore, Dual-tree representation is considered an important approach in this paper.

Traditional DTGP is implemented based on two key assumptions: (1) Two trees in an individual (i.e., VMSR and TSR) have the same impact on the whole individual's performance; and (2) all terminals are equally important while generating the initial trees and the mutated subtree. However, several recent studies clearly showed that different trees or terminals have significantly varied impacts on individual's performance/fitness [5], [18], [19]. In order to evolve offspring with good fitness, it is essential to explicitly consider the influence of each tree and each terminal on the performance of an evolved scheduling heuristic. Compared to crossover, it is more straightforward to achieve this goal through the mutation operator by selecting a tree with biased tree probabilities and replacing part of that tree with a subtree generated with biased terminal probabilities. Furthermore, in a preliminary work [5], we found that the crossover operator has less chance of generating more effective offspring for the DWS problems than other problems (empirical evidence can be found in Supplementary-A). Therefore, the mutation operator has attracted our attention to evolve high-quality offspring by adaptively enhancing the evolution chance of important trees and terminals.

Although several works in literature have raised approaches to consider the influence of trees and terminals, certain gaps persist that necessitate additional research efforts. In [19], a probabilistic adjustment at the tree selection level is proposed. However, this method is designed for transfer learning and cannot be used directly. At the terminal level, existing studies [18], [20] adjust the terminal probabilities based on individual fitnesses and terminal frequencies rather than the mathematical optimization approach.

In this paper, we propose to adaptively choose trees according to their performance impact and mutate the chosen tree with the tendency of using important terminals. For this purpose, we introduce three novel probability vectors, i.e., a *tree selection probability* vector (TSP) denoted as $\vec{ts}$, a *terminal weighting probability* vector for VMSR (TWPV) denoted as $\vec{tw}_{vmsr}$ and a *terminal weighting probability* vector for TSR (TWPT) denoted as $\vec{tw}_{tsr}$. As explained in Section IV, the three vectors are iteratively updated to support adaptive mutation, enabling DTGP to enhance the quality of its mutated offspring. Specifically, $\vec{ts}$ is used for stochastically selecting the tree to be mutated. Meanwhile, $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$ control terminal sampling upon constructing mutated subtrees.

We develop two new methods to adaptively update $\vec{ts}$, $\vec{tw}_{vmsr}$, and $\vec{tw}_{tsr}$. Specifically, $\vec{ts}$ is derived from an underlying Beta distribution, which is updated by using the 1/5 rule [21]. The *1/5 rule* is a technique originally proposed to enhance Evolution Strategy (ES) algorithms. It requires that 20% of mutations should lead to improvements. This is achieved by adjusting the *mutation deviation* during the evolutionary process, which measures the difference in terms of the average fitness before and after mutation. Moreover, $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$ are updated using the maximum likelihood estimation (MLE) technique [22] and convex optimization methods [23].

Driven by the new adaptive methods, we propose a new *DTGP algorithm with adaptive mutation* (DTGPAM) in this paper to jointly evolve effective VMSR and TSR for DCDWSC. Our new algorithm development gives rise to the following contributions:

1) We propose a new adaptive tree selection method to adjust $\vec{ts}$, which speeds up the convergence rate of DTGP by explicitly controlling the mutation deviation introduced n Section IV based on the 1/5 rule [21].
2) We propose a novel terminal weighting method that follows the maximum-likelihood principle to precisely calculate $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$. This method is capable of determining the terminal weights with respect to arbitrary terminal sampling ratio.
3) Through comprehensive experimental study on a wide range of DCDWSC problem instances, we show that the scheduling rules evolved by DTGPAM can outperform several state-of-the-art approaches under multiple different deadline constraints. DTGPAM also shows better convergence rates than other GP-based competing algorithms.
4) We perform further analysis on the three probability vectors, $\vec{ts}$, $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$. This analysis indicates the effectiveness of the two newly developed adaptive mechanisms.

II. Related Work

This section investigates the related works in terms of existing methods for DWS, multi-tree genetic programming, and adaptive mutation in genetic programming.

A. Existing Methods for DWS

To solve DWS, *scheduling heuristics/rules* are popularly used to schedule the execution of dynamic workflows in real-time, such as FCFS [9], MAXMIN and MINMIN [10]. They are efficient choices and are easy to implement. However, manually designing heuristics is time-consuming and requires domain knowledge. The designed heuristics may not perform reliably well whenever problem instances change.

To automatically design heuristics without domain knowledge, *hyper-heuristics* have been used to generate heuristics for job shop scheduling (JSS) [11], resource allocation [12], and arc routing problems [13]. Some recent studies attempted to develop GPHH algorithms to evolve heuristics for workflow scheduling in cloud [5], [14], [17].

As far as we are aware, most of the existing GPHH approaches for workflow scheduling problems focus primarily on *VM selection* [15], [16]. For example, a SGP approach in [16] was proposed to prioritize available VM instances for each task sequentially and assign it to the highest VM instance. In [15], a High-Level Heuristic SGP method was designed for DWS to prioritize the task-VM pair.

However, *task selection* is also important. Due to this reason, Xiao *et. al.* [14] designed a CCGP approach to evolve two rules under a cooperative coevolution framework. Experimentally, CCGP is shown to outperform several list-based heuristics and meta-heuristics.

While showing promise, CCGP did not explicitly handle the inter-dependencies between the rule pairs. To address this issue, in our preliminary work [5], we propose to solve the DCDWSC problem by developing a Dual-tree GP (DTGP) algorithm to evolve a combination of VMSR and TSR. We found through extensive experiments that DTGP can outperform both CCGP and SGP in most DCDWSC problem instances, confirming that it is effective to evolve VSMR and TSR in the form of a dual-tree representation. Furthermore, using both VSMR and TSR achieved better performance than using VMSR alone.

B. Multi-Tree Genetic Programming

In recent years, some researchers have begun to pay attention to the research of MTGP [19], [24]. It is a variant of GP that allows for the evolution of multiple interconnected trees. A special case where an individual consists of two trees is called DTGP.

Typically, genetic operators are applied to one tree randomly selected from each individual. Some studies have improved the genetic operators at the tree selection level. In [24], the crossover operator of DTGP was changed to mate one tree and swap another tree between the parent individuals. Al-Helali *et. al.* [19] proposed new probabilistic crossover and mutation operators for MTGP at the tree selection level to address transfer learning problems. According to self-defined probability functions, well-evolved trees in the individuals are more likely to be mated while poorly-evolved trees have more probability to be mutated. In [25], the *same-index crossover* operator, which is always performed on trees at the same index in each individual, was performed on all trees in an individual.

In summary, the selection of trees for evolution in MTGP poses a question that merits careful consideration. An existing study [19] proposed to set different selection probabilities based on a manually-defined function. However, it cannot be directly used to solve our problem, because they focus on the level of difference across different domains, rather than the improvement achieved by the mutated offspring. Different from [19], we propose to use the 1/5 rule to adaptively adjust the probabilities of selecting different trees based on the successful ratio of mutation.

C. Adaptive Mutation in GP

Mutation is an essential genetic operator for GP [26] to improve the quality of its evolved offspring. Existing works in the literature improve the effectiveness of mutation via adaptive mutation rate adjustment [27]–[29], terminal selection [18], [30], and terminal weighting [20], [31].

Adaptive mutation rate adjustment is achieved by changing the mutation rate during the evolutionary process based on several different factors, such as generation number [27], [29] and elite individual [28]. For example, an adaptive mutation rate that decreases with the generation number was designed for GP in [27]. This method gradually reduces the search space of mutation as the evolution process continues.

Terminal selection is used to mutate GP trees with a subset of terminals [18], [20], [30], assuming that individuals with good performance only rely on useful terminals. Zhang *et. al.* [18] proposed a terminal selection method for CCGP to improve the effectiveness and interpretability of the evolved rules/trees. Particularly, a terminal subset is determined first at the 50-th generation, and then only this terminal subset is used to generate mutated subtrees for the next 50 generations. The performance of this method is heavily influenced by the terminal subset selected based on the initial 50 generations.

Terminal weighting is designed with the belief that some terminals are more important and their use should be encouraged in mutation [20], [31]. Different from the terminal selection approach, this method focuses on assigning varied weights to different terminals while keeping the terminal set intact. In [31], a terminal's weight depends intuitively on the fitness of the corresponding individuals that use the terminal.

Since this paper aims to consider the influence of terminals, the terminal weighting method is adopted. Most existing works on terminal weighting rely on manually designed probability functions [18], [20], [31] according to terminal frequency and fitness, without utilizing any mathematically rigorous optimization methods. The maximum-likelihood principle is a widely-used statistical method that aims to estimate the parameters of a model in such a way that the observed data is most probable under that model [22]. Therefore, the maximum-likelihood principle is employed in this paper to accurately quantify the influence of terminals. By using this technique, we can accurately quantify the terminal influence to maximize the likelihood of generating high-quality offspring.

III. Preliminary

This section formulates the *deadline-constrained dynamic workflow scheduling in cloud (DCDWSC) problem* based on existing works [3], [5], [8], [32]. The relevant parameters are listed in Table I.

TABLE I
NOMENCLATURE

Notations	Descriptions
W_i	The i-th workflow
V_k	The k-th VM instance
VT_l	The l-th VM type
AT_i	Arrival time of workflow W_i
DL_i	Deadline of workflow W_i
t_{ij}	The j-th task in workflow W_i
NOC_{ij}	Number of successors of task t_{ij} in W_i
TS_{ij}	Task size of t_{ij}
CU_k	Compute unit of V_k
$PRICE_k$	Hourly rental fee of V_k
ET_{ij}^k	Execution time of t_{ij} on V_k
LFT_{ij}^k	Latest finish time of t_{ij} on V_k at the current time
RWT_{ij}^k	Relative waiting time of t_{ij} on VM queue of V_k
WT_{ij}^k	Waiting time of t_{ij} on V_k
ST_{ij}^k	Start time of t_{ij} on V_k
FT_{ij}^k	Finish time of t_{ij} on V_k
NOR_i	Number of unassigned tasks remaining in W_i
RDL_i	Remaining time of W_i before the DL_i is reached, calculated by $RDL_i = DL_i - current\_time$
RT_{ij}	Ready time of task t_{ij}
NIQ_k	Number of pending tasks in VM queue of V_k
TIQ_k	Total execution time of pending tasks in VM queue of V_k
VMR_k	Remaining rental period of V_k of its current lease

In dynamic workflow scheduling problems, a set of dynamically arriving workflows $\mathcal{W} = \{W_1, W_2, \cdots, W_m\}$ needs to be executed by a time-varying set of VM instances $\mathcal{V} = \{V_1, V_2, \cdots, V_n\}$ rented from a predetermined set of VM types $\mathcal{VT} = \{VT_1, VT_2, \cdots, VT_u\}$. Information regarding any workflow, such as workflow size and workflow pattern, remains unknown until it arrives. Each workflow W_i is modeled as a *directed acyclic graph* $DAG_i \in \{T_i, E_i\}$, where *nodes* represent tasks $T_i = \{t_{i1}, t_{i2}, \cdots, t_{iq_i}\}$ and *edges* connect a predecessor task to one of its successor tasks to enforce their execution order $E_i = \{(t_{i1}, t_{i2}), (t_{i1}, t_{i3}), \cdots, (t_{ij}, t_{iq_i})\}$, as shown in Fig. 2.

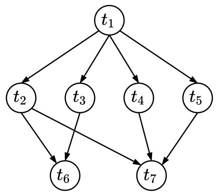


Fig. 2. An example of a workflow modeled as a DAG.

Each workflow W_i has an arrival time AT_i and a deadline DL_i. For each task t_{ij} in workflow W_i, let NOC_{ij} denote the number of its successor tasks in the workflow and TS_{ij} denote its task size. Each VM instance V_k has several attributes: its compute unit (i.e., computation capacity) CU_k, its memory size MEM_k and its hourly rental fee $PRICE_k$.

During the scheduling process, there are some constraints in the DCDWSC problem that needs to be clarified:

- Only tasks in the ready state (explained in Eq. (1)) can be scheduled.
- A task can only be processed by one of the available VM instances and cannot be interrupted during its execution.
- A VM instance can only handle one task at one time.
- Every workflow is associated with a deadline that needs to be met, otherwise the broker will compensate its users.
- VM instances are rented on an hourly basis. One additional hour of lease will be automatically applied to a cloud resource whenever its current lease hour is exceeded.

The relationship of the relevant attributes is illustrated in Fig. 3 and their definitions are described as follows.

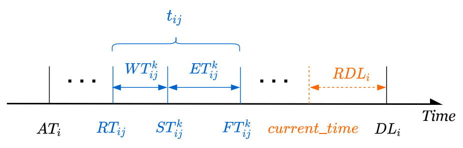


Fig. 3. Attribute relationship of task t_{ij} in workflow W_i.

An unprocessed task transforms into a *ready* task immediately when all its predecessor tasks in the same workflow have been processed. Thus, the *ready time* RT_{ij} of task t_{ij} can be calculated by

$$RT_{ij} = \begin{cases} AT_i, & if\ task_{ij}\ is\ an\ entry\ task \\ \max_{z \in pred(t_{ij})} \{FT_{iz}\}, & otherwise \end{cases} \quad (1)$$

where AT_i is the arrival time of workflow W_i.

Let ET_{ij}^k denote the *execution time* of task t_{ij} on VM instance V_k. It can be calculated by

$$ET_{ij}^k = \frac{TS_{ij}}{CU_k} \quad (2)$$

Note that the higher the computer performance CU_k of the selected VM instance V_k, the shorter the execution time.

Let LFT_{ij}^k denote the *latest finish time* of t_{ij} on VM instance V_k at the moment to be allocated. It can be calculated by

$$LFT_{ij}^k = TIQ_k + ET_{ij}^k \quad (3)$$

meaning that t_{ij} will be processed last by V_k.

Let RWT_{ij}^k denote the *relative waiting time* of task t_{ij} on the VM queue of V_k at the current time. It is calculated by

$$RWT_{ij}^k = current\_time - RT_{ij} \quad (4)$$

Note that task t_{ij} has not been processed at $current\_time$.

Let FT_{ij}^k denote the actual *finish time* of task t_{ij} on VM instance V_k. It depends on its actual start time ST_{ij}^k on V_k and can be calculated by

$$FT_{ij}^k = ST_{ij}^k + ET_{ij}^k \quad (5)$$

Let WT_{ij}^k denote the *waiting time* of task t_{ij} on VM instance V_k, i.e., the time interval between the actual start time ST_{ij}^k of task t_{ij} on V_k and its ready time RT_{ij} (regardless of which machine). It can be calculated by

$$WT_{ij}^k = ST_{ij}^k - RT_{ij} \quad (6)$$

Let $RentFee_k$ denote the *rental fee* of VM instance V_k. It is determined by the rental time and corresponding hourly prices, and can be calculated by

$$RentFee_k = PRICE_k \cdot \left\lceil \frac{FT_{t_{last}}^k - ST_{t_{first}}^k}{3600} \right\rceil \quad (7)$$

where t_{first} and t_{last} are the first and the last tasks processed on VM instance V_k, thus the right part of Eq. (7) represents the lease time of V_k measured in hours.

According to [2], if a workflow W_i cannot be completed before its deadline, a penalty is applied based on a predefined penalty rate. Let $Penalty_i$ denote the penalty of workflow W_i which can calculated by

$$Penalty_i = \delta \cdot \max\{0, AT_i + Makespan_i - DL_i\} \tag{8}$$

$$DL_i = AT_i + \xi \cdot MinMakespan_i \tag{9}$$

where δ is the *penalty coefficient* [33], and a bigger the value of δ represents a lower tolerance for violating workflow deadline. In Eq. (9), ξ is a *deadline relaxation coefficient* [2], and a larger value indicates a looser constraint. Moreover, $MinMakespan_i$ relates to the shortest execution time of W_i in theory, calculated by processing all tasks in W_i on the fastest VM instances without any delay.

The *objective* of the DCDWSC problem is to minimize the *total cost* incurred for workflow execution, including *VM rental fees* of all VM instances and *workflow deadline penalties* triggered by violations, as following

$$\min TotalCost = \sum_{k\in RVMS} RentFee_k + \sum_{i:W_i\in\mathcal{W}} Penalty_i \tag{10}$$

where *RVMS* is the set of rental VM instances, and $\mathcal{W}$ is the set of all workflows arrived within a certain period of time. More details are defined below.

There are two decision variables in the DCDWSC problem, i.e., a *VM selection rule* (VMSR) and a *task selection rule* (TSR). These are related to two types of scheduling decisions, VM selection and task selection, each supported by a heuristic referring to Eq. (11) and Eq. (12).

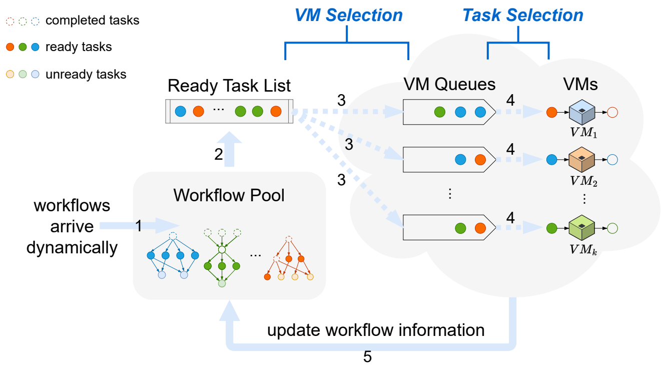


Fig. 4. Example of scheduling workflows with a scheduling heuristic.

Fig. 4 illustrates the overall scheduling process of DCDWSC with the support of a VMSR and a TSR. The scheduling process starts from *label 1*, where a large number of workflows with heterogeneous patterns and sizes are dynamically submitted to the workflow pool. The broker then identifies all ready tasks (explained in Section III) and enters them into the *ready task list* (*label 2*). Tasks in the *ready task list* (RTL) are sorted sequentially by RT_{ij}, or by the number of successor tasks NOC_{ij} if their RT_{ij} are the same, otherwise by their system indexes. Next, tasks in the ready task list will be sequentially allocated to one of the available VM instance candidates by a VM selection method π_{vmsr} (*label 3*). Whenever the leased VM instance is idle, it will select a pending task from its VM queue for execution by a task selection method π_{tsr} (*label 4*). Continuously, task completion information based on current VM instances is fed back to the workflow pool in real-time (*label 5*) to activate the remaining unassigned tasks in workflows to enter the ready task list.

Particularly, *VM selection* in *label 3* refers to selecting a VM instance (e.g., V_k) with the best priority value from all VM instance candidates according to a *VMSR* (represented as π_{vmsr}) for the first ready task t_{ij} in the ready task list, formulated by

$$V_k = \operatorname*{arg\,min}_{\theta\in VMcandidates} \pi_{vmsr}(t_{ij}, V_\theta) \tag{11}$$

Task selection in *label 4* refers to selecting a pending task (e.g., t_n) with the best priority value from the VM queue of a leased VM instance V_k to execute based on a *TSR* (represented as π_{tsr}), formulated by

$$t_{ij} = \operatorname*{arg\,min}_{\eta\in VMqueue_k} \pi_{tsr}(t_\eta, V_k) \tag{12}$$

IV. Proposed Algorithm

This section proposes the *Dual-tree Genetic Programming with Adaptive Mutation* (DTGPAM) algorithm for solving the DCDWSC problem. The algorithmic framework of DTGPAM is first introduced, followed by the development of two adaptive methods respectively for tree selection and terminal weighting. They together enable us to develop a new adaptive mutation method to effectively evolve scheduling heuristics.

A. Framework of the DTGPAM Algorithm

The overall framework of DTGPAM is shown in Fig. 5. An initial population is composed of randomly generated individuals. The representation of each individual adopts a tree-based structure, consisting of a pair of rules (i.e., VMSR and TSR), as shown in Fig. 6. Each individual will be evaluated using a fitness function. Subsequently, parent selection is applied via the tournament selection method. The selected parent individuals are evolved by reproduction, elitism, crossover and mutation operators to produce their offspring. The above steps are repeated until the termination criteria is met. Eventually, the fittest individual is obtained as the best scheduling rule/heuristic.

As indicated in Fig. 5, *Adaptive Mutation* is a key technical novelty of DTGPAM. Its operation depends on three newly proposed probability vectors $\vec{ts}$, $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$. Details of the Adaptive Mutation and the three vectors will be described in the subsequent subsections.

B. Adaptive Mutation via Three Probability Vectors

This subsection explains how to use three probability vectors to improve the quality of evolved VMSRs and TSRs.

To achieve adaptive mutation, we first define three important probability vectors: $\vec{ts}$, $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$. The *tree selection probability vector* $\vec{ts}$ is defined as

$$\vec{ts} = [p_{both}, p_{vmsr}, p_{tsr}]^T \tag{13}$$

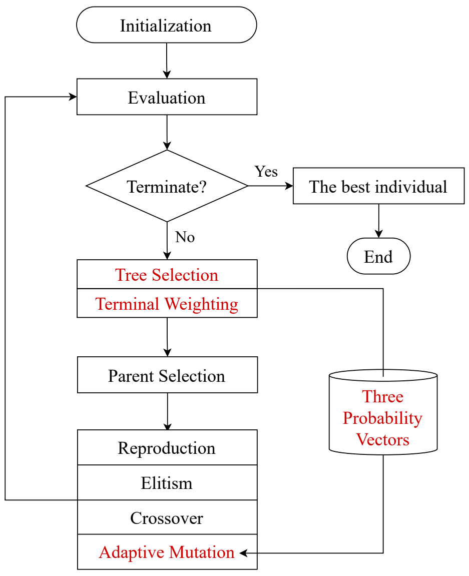


Fig. 5. Flowchart of the DTGPAM algorithm.

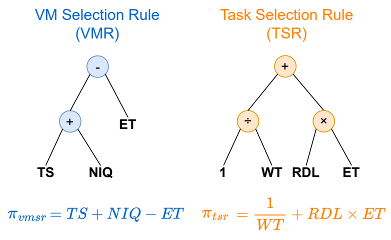

VM Selection Rule (VMR)
Task Selection Rule (TSR)
-
+
ET
TS
NIQ
+
÷
×
1
WT
RDL
ET
$\pi_{vmsr} = TS + NIQ - ET$
$\pi_{tsr} = \frac{1}{WT} + RDL \times ET$

Fig. 6. Example of a scheduling heuristic.

where $0 \leq p_{both}, p_{vmsr}, p_{tsr} \leq 1$ and $p_{both}+p_{vmsr}+p_{tsr} = 1$. p_{both}, p_{vmsr}, and p_{tsr} indicate respectively the probability for selecting both trees, VMSR or TSR for mutation, as shown in Fig. 6. The *terminal weighting probability vectors* $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$ are defined as

$$\begin{aligned}\vec{tw}_{vmsr} =& [p_1, \cdots, p_j, \cdots, p_{v_1}]^T \\ \vec{tw}_{tsr} =& [\hat{p}_1, \cdots, \hat{p}_j, \cdots, \hat{p}_{v_2}]^T\end{aligned} \tag{14}$$

Each terminal type can have multiple terminals. Let p_j denotes the probability for selecting the j-th terminal type while building a mutated subtree for VMSR, and $\hat{p}_j$ refers to the probability for selecting the j-th terminal type while building a mutated subtree for TSR. We require $0 \leq p_j, \hat{p}_j \leq 1$, $\sum_{j=1}^{v_1} p_j = 1$ and $\sum_{j=1}^{v_2} \hat{p}_j = 1$.

The details of the proposed adaptive mutation method are described in Algorithm 1. To begin with, one or both trees in individual Ind are probabilistically selected according to $\vec{ts}$ (*line 1*). For each selected tree $tree$, a random subtree $subtree_{pre}$ is removed (*line 3*). Subject to the selected tree, either $\vec{tw}_{vmsr}$ or $\vec{tw}_{tsr}$ (represented as tw_s) are used to create the mutated substree $subtree_{new}$ at *line 9*. To encourage exploration, we also adopt a sampling ratio r to allow every terminal to be selected uniformly at random with a small probability. Finally, $subtree_{pre}$ in $tree$ is replaced by $subtree_{new}$ (*line 5*) to form a mutated individual Ind' (*line 6*).

The method of generating a new subtree via $\vec{tw}_s$ (i.e., $\vec{tw}_{vmsr} or \vec{tw}_{tsr}$) and r is elaborated from *line 9* to *line 21*.

Algorithm 1: Adaptive Mutation of DTGPAM

```
Input: An individual $Ind$, $\vec{ts}$, $\vec{tw}_{vmsr}$, $\vec{tw}_{tsr}$, $r$, $tree\_depth$
Output: A mutated individual $Ind$
1  $treeSet \leftarrow$ Select $VMSR, TSR$ or $both$ from $Ind$ with probabilities in $\vec{ts}$
2  for $\forall$ $tree \in treeSet$ do
3      Randomly select a $subtree_{pre}$ in $tree$
4      $subtree_{new} \leftarrow$ generate_subtree($tree\_depth$, $\vec{tw}_s$, r)
5      $tree' \leftarrow$ Replace $subtree_{pre}$ by $subtree_{new}$
6      $Ind[tree] \leftarrow tree'$
7  end
8  return $Ind$
9  Function generate_subtree($tree\_depth$, $\vec{tw}_s$, r):
10     while $|$ $subtree_{new}$ $| \leq tree\_depth$ do
11         if $must\ be\ a\ terminal$ then
12             if $random.() < r$ then
                   // Probability
13                 $subtree_{new} \leftarrow subtree_{new} +$ a terminal node sampled probabilistically according to $\vec{tw}_s$
14             else
                   // Random
15                 $subtree_{new} \leftarrow subtree_{new} +$ a terminal node sampled uniformly at random
16             end
17         else
18             $subtree_{new} \leftarrow subtree_{new} +$ a randomly sampled function node
19         end
20     end
21     return $subtree_{new}$
```

Specifically, the sampling ratio $r \in (0, 1]$ is used to determine whether a terminal should be sampled accordingly to tw or uniformly at random (*line 12*). In the former case (*line 13*), terminals with higher weights are more likely to be selected. In the latter case (*line 15*), all terminals have an equal probability of being selected.

Algorithm 1 allows r to be adaptively adjusted by a cooling schedule function [34], which is a crucial component of simulated annealing algorithms used for trade-off exploration and exploitation. Three widely used cooling functions, i.e., logarithmical multiplicative cooling, exponential multiplicative cooling, and linear adaptive cooling, are often considered in literature [35], [36]. Experimental studies will be presented in Subsection VI-B to choose an appropriate cooling schedule function.

C. Algorithm for Adaptive Updating of $\vec{ts}$

This subsection proposes a method based on the 1/5 rule to adaptively update $\vec{ts}$ based on the *mutation success ratio* (see Eq. (18)) so as to improve the convergence speed. Specifically, $\vec{ts}$ is determined by an underlying Beta distribution parameterized by α, which is adjusted by the 1/5 rule [21].

To understand our adaptive method for updating $\vec{ts}$, we need to tackle two problems. The *first problem* is to how to build $\vec{ts}$. We propose to build $\vec{ts}$ based on a Beta distribution that comprises of two fixed parameters (i.e., β and $points$) and one varying parameter (i.e., α).

We choose the Beta distribution due to the following advantages. Firstly, $\vec{ts}$ defined in this way is guaranteed to provide a valid probability distribution, i.e., all its elements add up to 1 and each element is non-negative. Secondly, the Beta distribution allows a smooth adjustment of three probabilities p_{both}, p_{vmsr} and p_{tsr} in $\vec{ts}$ by controlling a single parameter α. Fig. 7 depicts multiple probability density functions $Pr_\beta(x;\alpha,\beta)$ of the Beta distribution with α ranging from 2 to 23 and a step size of 3.

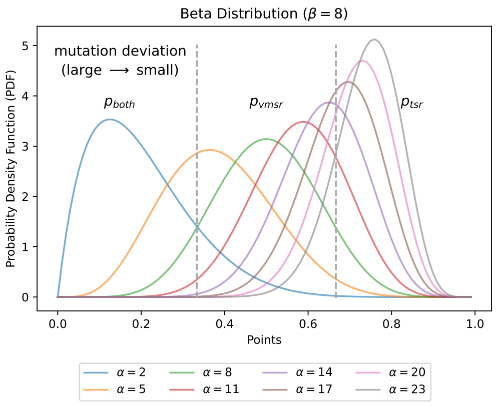


Fig. 7. Example of obtaining a vector by Beta distribution.

We divide the definitional domain of the Beta distribution [0,1] into three equal intervals based on the two thresholds, i.e. $x_1 = 0.33$ and $x_2 = 0.66$. Consequently, three areas are obtained corresponding to the values of $p_{both}, p_{vmsr}, p_{tsr}$, calculated by

$$\begin{aligned} p_{both} &= \int_0^{0.33} Pr_\beta(x;\alpha,\beta)\mathrm{d}x \\ p_{vmsr} &= \int_{0.33}^{0.66} Pr_\beta(x;\alpha,\beta)\mathrm{d}x \\ p_{tsr} &= \int_{0.66}^{1} Pr_\beta(x;\alpha,\beta)\mathrm{d}x \end{aligned} \tag{15}$$

After some empirical studies (see Supplementary-D), we set $\beta = 8$ to to avoid significant differences among the three probability values. Since the maximum value of p_{vmsr} is bounded approximately by 0.85, a limit of $\alpha \in [2, 23]$ is set to bound the maximum values of p_{both} and p_{tsr} around 0.85 to ensure fairness.

In this paper, we introduce a metric, *mutation deviation*, for measuring the fitness variability of the mutated individual, formulated by

$$\triangle_{mut} = \frac{1}{n}\sum_{i=1}^{n}\left|fitness_i' - fitness_i\right| \tag{16}$$

where $fitness_i'$ is the fitness of the mutated individual calculated by Eq. (29). and n is the number of individuals experiencing mutation within a population.

We expect that a larger α leads to a smaller *mutation deviation*, since TSR was observed to have a smaller impact on fitness than VMSR in [5]. As shown in Fig. 7, the *mode* of the probability density function shifts from left to right as α increases. Accordingly, the value of p_{tsr} becomes larger. As an example, when α goes from 8 to 11, the value of $\vec{ts}$ shifts from [0.09, 0.82, 0.10] to [0.01, 0.76, 0.22]. Therefore, the *mutation deviation* is expected to be smaller. This assumption will be verified in Subsection VI-F.

The *second problem* is how to update $\vec{ts}$ so as to adaptively change the probabilities of mutating VMSR and TSR of any evolved scheduling heuristic. We propose to tackle this problem through updating parameter α of the Beta distribution by utilizing the 1/5 rule [21]. According to this rule, the *success rate* for mutation should be maintained at the level of approximately 1/5 for effective evolution with solid mathematical guarantees [37]. A mutation is considered successful if the mutation leads to an improvement in fitness.

In this paper, following existing works [37]–[39], we keep track of the *mutation success ratio* $\bar{\phi}$ in each generation by using

$$\phi_{cur} = \frac{\text{number of successful mutations}}{\text{number of total mutations}} \tag{17}$$

$$\bar{\phi}_i = \bar{\phi}_{i-1} + s_{ema} \cdot (\phi_{cur} - \bar{\phi}_{i-1}) \tag{18}$$

where ϕ_{cur} refers to the percentage of successful mutations observed in the i-th generation. $\bar{\phi}_i$ is the estimated mutation success ratio of the i-th generation. Moreover, s_{ema} is the smoothing factor. Its recommended value is determined as: $2/(10+1) = 0.1818$ [40].

Algorithm 2: `TreeSelection`$(\bar{\phi}, G, \alpha)$

```
Input: Successful ratio $\bar{\phi}$, a moving window length $G$, $\alpha$ in Beta
       Distribution
Output: A tree selection probability vector $\vec{ts}$
1 c = 0.817, $\beta = 8$, points = $[0.33, 0.66]$
2 if $\bar{\phi} < 1/5$ then
3 |   $\alpha \leftarrow \alpha / c$
4 else if $\bar{\phi} > 1/5$ then
5 |   $\alpha \leftarrow \alpha \cdot c$
6 end
7 $\vec{ts} \leftarrow Beta(\alpha, \beta, points)$
8 return $\vec{ts}$
```

Obeying the 1/5 rule, we develop Algorithm 2 to update $\vec{ts}$. The parameter c adopts the commonly used value of 0.817 for the 1/5 rule [21], [37]. As described in the algorithm pseudo-code, if $\bar{\phi}$ is less than 1/5 (*line 3*), then the mutation introduces too much exploration, which leads to large but infrequent improvements, thus the mutation deviation should be decreased (i.e., increase α to move the *mode* of the probability density function to the right in Fig. 7). If $\bar{\phi}$ is more than 1/5 (*line 5*), then the mutation introduces too little exploration, which leads to small improvements, thus the mutation deviation should be increased (i.e., decrease α to move the *mode* of the probability density function to the left in Fig. 7).

D. Algorithm for Computing $\vec{tw}_{vmsr}$ and $\vec{tw}_{tsr}$

This subsection proposes a new general algorithm to compute terminal weights for either $\vec{tw}_{vmsr}$ or $\vec{tw}_{tsr}$ based on high-performing individuals in each generation, as elaborated in Algorithm 3. For the sake of simplicity, we use the joint notation $\vec{tw}$ to represent either $\vec{tw}_{vmsr}$ or $\vec{tw}_{tsr}$ as both tree types use different terminal sets and need to be calculated separately in the same way. The updated $\vec{tw}$ is expected to guide mutation to focus more on important terminals. The maximum likelihood estimation (MLE) method [22] is adopted to formulate a mathematical optimization problem, which is solved by the convex optimization and water-filling techniques to obtain $\vec{tw}$.

Algorithm 3 is designed to calculate $\vec{tw}$ in consideration of both the adaptive sampling ratio r and the MLE. Assuming that there are m high-fitness trees that contain n terminals in the terminal set, then x_{ji} represents the number of the i-th terminal type found in $tree_j$.

```
Algorithm 3: TerminalWeighting($P$)
   Input: A population $P$
   Output: A feature weighting probability vector $\vec{tw}$
1  $P_{tw}$ ← Select the best $n_{tw}$ individuals from $P$
2  $\chi$ ← Count the terminal frequency matrix of $P_{tw}$
3  $\ln \mathcal{P}$ ← Formulate the log-likelihood function via $\chi$
4  $\vec{tw}$ ← Solve $\ln \mathcal{P}$ by convex optimization techniques
5  return $\vec{tw}$
```

The maximum likelihood function $\mathcal{P}$ is formulated by

$$\mathcal{P} = \prod_{j=1}^{m} P(tree_j) \tag{19}$$

where $P(tree_j)$ represents the probability of selecting several terminal nodes from the terminal set to construct the specific tree $tree_j$. It can be calculated based on the j-th row of χ using the formula below:

$$P(tree_j) = \prod_{i=1}^{n} ((1-r) \cdot \frac{1}{n} + r \cdot p_i)^{x_{ji}} \tag{20}$$

where $r \cdot p_j$ indicates the probability that the j-th terminal type was sampled according to $\vec{tw}$, and $(1-r)/n$ indicates the probability that the j-th terminal type was sampled uniformly at random.

MLE requires us to find $\vec{tw}$ that maximizes $\mathcal{P}$ in Eq. (19). Note that maximizing $\ln \mathcal{P}$ in Eq. (21) is equivalent to maximizing $\mathcal{P}$ directly.

$$\begin{aligned} \ln \mathcal{P} &= \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ji} \cdot \ln (r \cdot p_i + \frac{1-r}{n}) \\ &= \sum_{i=1}^{n} x_i \cdot \ln (r \cdot p_i + \frac{1-r}{n}) \end{aligned} \tag{21}$$

where $x_i = \sum_{j=1}^{m} x_{ji}$ is the i-th column of χ.

The problem in Eq. (21) is therefore transformed into an equivalent problem of maximizing the log-likelihood function, formulated by

$$minimum \quad f(p) = -\sum_{i=1}^{n} x_i \cdot \ln (r \cdot p_i + \frac{1-r}{n}) \tag{22}$$

$$s.t. \quad 0 \leq p_i \leq 1 \quad (i = 1, \cdots, n) \tag{22a}$$

$$\sum_{i=1}^{n} p_i = 1 \tag{22b}$$

We must solve Eq. (22) to obtain $\vec{tw}$. Since Eq. (22) is a convex optimization problem (see Supplementary-B), its solution must satisfy the Lagrange Karush-Kuhn-Tucker (KKT) conditions [23] presented below:

$$\begin{aligned} L(p_i, \lambda, \nu) = &-\sum_{i=1}^{n} x_i \cdot \ln (r \cdot p_i + \frac{1-r}{n}) \\ &+ \sum_{i=1}^{n} \lambda_i(-p_i) + \nu(\sum_{i=1}^{n} p_i - 1) \end{aligned} \tag{23}$$

where λ_i is the Lagrange multiplier associated with the i-th inequality constraint in Eq. (22a). ν is the Lagrange multiplier associated with the equality constraint in Eq. (22b).

According to the Lagrange KKT conditions [23], we have

$$\begin{aligned} & p_i^* \geq 0 \quad (i = 1, \cdots, n) \\ & \sum_{i=1}^{n} p_i^* - 1 = 0 \\ & \lambda^* \geq 0 \\ & \lambda_i^* \cdot p_i^* = 0 \quad (i = 1, \cdots, n) \\ & \frac{-r \cdot x_i}{r \cdot p_i + \frac{1-r}{n}} - \lambda_i^* + \nu^* = 0 \end{aligned} \tag{24}$$

λ^* in Eq. (24) can be eliminated first, leaving

$$p_i^* = \begin{cases} \frac{x_i}{\nu^*} - \frac{1-r}{nr}, & \nu^* < \frac{nr \cdot x_i}{1-r} \\ 0, & \nu^* \geq \frac{nr \cdot x_i}{1-r} \end{cases} \tag{25}$$

An alternative form of the above equation is

$$p_i^* = max \left\{ 0, \frac{x_i}{\nu^*} - \frac{1-r}{nr} \right\} \tag{26}$$

Substituting this expression for p_i^* into the condition in Eq. (22a), we obtain

$$\sum_{i=1}^{n} max \left\{ 0, \frac{x_i}{\nu^*} - \frac{1-r}{nr} \right\} = 1 \tag{27}$$

Since Eq. (27) may not have an analytical solution, the *water-filling* algorithm [41] is used to efficiently obtain the empirical solution of Eq. (27) and finally the value of p_j^*. The computational complexity of the water filling algorithm is $O(n \log n)$, where n is the size of the terminal set [41]. When n is not large as for our DCDWSC problem, the water-filling algorithm can be used efficiently. The validity of Eq. (27) is verified in Supplementary-C. Furthermore, we conduct a detailed theoretical analysis on the effectiveness of the proposed MLE-based method in Supplementary-F.

V. Experiment Design

The simulation configuration, baseline approaches and algorithm setting are introduced in this section.

A. Simulation Configuration

Simulation is used to measure the performance of scheduling heuristics on the DCDWSC problem. Our simulated DCDWSC problem instances consist of the following components.

VM Types: Table II summarizes the configurations of 6 different VM types. We configure all experimented VM types according to Amazon EC2[1]. Meanwhile, we impose no limit on the maximum number of instances of each VM type.

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

TABLE II
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

Workflow Sets: We design problem instances with different complexities, i.e., workflow patterns and task numbers. Table III shows 3 workflow sets (i.e., *Mix_Small*, *Mix_Middle* and *Mix_Large*) used for training and testing scheduling heuristics, consisting of 4 popular workflow patterns[2] (i.e., CyberShake, Montage,, Inspiral and SIPHT) with different task numbers (e.g., 30, 50, and 100).

TABLE III
THE COMPONENTS OF THREE WORKFLOW SETS

Workflow Set	Component
Mix_Small	CyberShake_30, Montage_25, Inspiral_30, Sipht_30
Mix_Middle	CyberShake_30, Montage_25, Inspiral_30, Sipht_30, CyberShake_50, Montage_50, Inspiral_50, Sipht_60
Mix_Large	CyberShake_30, Montage_25, Inspiral_30, Sipht_30, CyberShake_50, Montage_50, Inspiral_50, Sipht_60, CyberShake_100, Montage_100, Inspiral_100, Sipht_100

Training Stage: Through the training stage, we can obtain the best GPHH heuristic with the highest fitness across multiple training problem instances. A *training problem instance* consists of 30 workflows randomly sampled from the *Mix_Small* workflow set. In each generation, every GP individual will be evaluated on 5 randomly generated training problem instances to calculate its fitness, following the settings used in existing studies [5], [15], [17], [42] and our preliminary study.

Testing Stage: Evolved heuristics will be further tested in three testing scenarios to compare its performance with competing algorithms summarized in Subsection VI-A. A *testing problem instances* consists of 30 workflows randomly sampled from the corresponding workflow set, i.e., *Mix_Small* (S for short), *Mix_Middle* (M for short) or *Mix_Large* (L for short) workflow set. A *testing scenario* consists of 30 randomly generated testing problem instances. Refer to Table VI, $\langle 12, S\rangle$ denotes that $\xi = 12$ and heuristics are tested on the *Mix_Small* scenario.

Request generation: Workflow requests are sent to the broker over time according to a Poisson distribution with $\lambda = 0.01$ [43]. In Eq. (9), the deadline relaxation coefficient is set to $\xi \in \{1, 6, 12, 18, 24\}$. As ξ increases, the deadline becomes more relaxed deadline and can be fulfilled by using cheaper VM instances. Moreover, the penalty coefficient in Eq. (8) is $\delta = \$0.24/h$ according to [33].

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

B. Baseline Approaches

Four competing approaches are listed below for comparison.

- $JIC\text{-}FCFS$: The VMSR proposed in [8] was used to first select available VM instances that do not cause the workflow to exceed its deadline, and then select the cheapest VM instance. In addition, FCFS [9] is used to select a task for any idle VM instances.
- $HEFT\text{-}FCFS$: This method uses the widely-used approach namely HEFT [6] to select VM instances with the earliest task finish time in conjunction with FCFS [9] for task selection.
- $DTGP$: This method evolves VMSR and TSR via standard mutation [5], i.e., either VMSR or TSR will be mutated with a randomly generated subtree.
- $DTGP^f$: We apply the terminal weighting method proposed for SGP in [31] to drive adaptive mutation in DTGP [5]. Following [31], terminal weights are calculated below in consideration of the frequency of each terminal type in every selected tree as well as the fitness of the respective trees ($fitness_j^{(i)}$):

$$weight_k = \sum_{i=0}^{gen} \sum_{j=0}^{pop} fitness_j^{(i)} \times \frac{num\_terminal_{k,j}}{total\_terminal_j} \quad (28)$$

  where the numerator denotes the number of the k-th terminal type found in $tree_j$ at the i-th generation, and the denominator denotes the total number of terminals found in $tree_j$ at the i-th generation.
- $DTGPAM$: The new DTGPAM algorithm presented in Section IV of this paper.

Each competing algorithm will be evaluated by performing 50 independent runs to determine its average total cost. A smaller total cost indicates a better performance. The Wilcoxon test with a significance level of 0.05 is adopted to compare all competing algorithms in terms of their average total cost. In Table VI, "+", "−" or "≈" indicate that the corresponding result is significantly better, worse or similar to its counterparts.

C. Algorithm Setting

Two terminal sets for building VMSRs and TSRs are summarized in Table IV. All terminals are categorized into task-related, VM-related, workflow-related, and problem-specific terminals based on the information captured by each terminal. Corresponding notations for each terminal have also been summarized in Table I. Following many existing works [5], [14], [15], we set $\{+, -, \times, \div, max, min\}$ as *function* nodes for constructing the GP trees.

Fitness evaluation: Each individual ind in the population will be evaluated on 5 workflow scheduling problem instances, each consisting of 30 workflows and 6 VM types, in order to accurately determine the practical performance of the rule pair. The average total cost formulated in Eq. (10) across all the n problem instances achieved by the individual is defined as its fitness value below:

$$fitness_{ind} = \frac{1}{5} \sum_{i=1}^{5} total\_cost(ind) \quad (29)$$

TABLE IV
THE TERMINAL SET OF VMSR AND TSR

	VMSR	**TSR**
task-related	TS	ET
	ET	RWT
VM-related	CU	TIQ
	$PRICE$	NIQ
	TIQ	
	VMR	
	LFT	
	NIQ	
workflow-related	NOC	NOC
	NOR	NOR
problem-specific	RDL	RDL

Parameter setting: Following [5], [31], three GP-based algorithms share several identical parameter settings. The population size is set to 1024, the number of generations is 51, the number of the elite is 1, and the tournament size is 7. The initial depth of the GP tree is in the range from 2 to 6, and the maximum depth of the GP tree during evolution is 8. The initial population is produced by the popular *ramped-half-and-half* method [14]. Besides these, other parameter settings are different among the three competing algorithms. They are summarized in Table V. The first three parameters are set following the experimental validation in [31]. Since the proposed mutation operator is adaptive, we set the fourth parameter as 0. The sensitivity analysis of the last two parameters is reported in Subsection VI-B.

TABLE V
PARAMETER SETTING FOR GP-BASED ALGORITHMS

Parameter	DTGP [5]	DTGP^f [5], [31]	DTGPAM
crossover rate	0.8	0.65	0.65
mutation rate	0.15	0.3	0.3
reproduction rate	0.05	0.05	0.05
starting of adaptive mutation	–	15	0
sampling ratio r	–	1.0	*log_rev*
initial α	–	–	5

* see Table VIII for *log_rev*.

Computation Complexity: The complexity of the adaptive mutation operator in DTGPAM across all generations includes three components [41], [44], [45]: (1) The complexity of replacing a subtree is $O(d_2 \log d_2 NG)$, where N is the population size, G is the number of generations, and d_2 is the maximum depth of generated subtrees in mutation; (2) The complexity of estimating tree selection probabilities is $O(NG)$; and (3) The complexity of estimating terminal weights is $O((l_1 N + s_1 \log s_1 + s_2 \log s_2)G)$, where l_1 is the maximum allowed number of nodes in a GP tree subject to the tree depth limit, s_1 and s_2 are the terminal set size of VMSR and TSR. In summary, although the adaptive mutation operator in DTGPAM exhibits higher complexity compared to the normal mutation operator in DTGP, the DTGPAM algorithm has the same overall complexity level as that of DTGP, i.e., $O((d_1 \log d_1 + s_k + l_1)NG)$ where s_k is the tournament size. (See Supplementary-E for the detailed process.)

VI. RESULTS AND DISCUSSIONS

A. Main Experiment Results

Table VI reports the mean (standard deviation) total cost achieved by JIC-FCFS, HEFT-FCFS, DTGP, DTGP^f and DTGPAM across 50 independent runs. In general, the total cost of all algorithms decreases upon increasing ξ, as they enable workflows with loose deadline constraints to be executed on cheaper VM instances. When $\xi = 1$, all algorithms prefer to choose the fast VM instance to prevent the deadline violation penalty, which leads to small differences among different algorithms.

We found that DTGPAM achieved the minimal total cost on half of testing scenarios and performed significantly better under loose (i.e., $\xi = 12, 18, 24$) deadlines. This demonstrates that DTGPAM is able to make better VM selection and task selection decisions than its counterparts when tasks have a low risk of expiration (i.e., loose deadline). Moreover, although the scheduling heuristic is trained on a set of small-sized workflows (i.e., *Mix_Small*) to save computational cost, it manages to achieve consistently good testing performance on large-sized workflows (e.g., $\langle 6, L\rangle$, $\langle 24, M\rangle$ and $\langle 24, L\rangle$). This demonstrates the good scalability of DTGPAM.

Although DTGPAM is not significantly better than DTGP and DTGP^f on testing scenarios with tight deadlines (i.e., ξ=$1, 6$), it still obtained lower average total costs. Comparing DTGP and DTGP^f, the results of DTGP^f are significantly better than DTGP in scenarios $\langle 18, S\rangle$ and $\langle 18, L\rangle$, and the average total cost of DTGP^f are less than that of DTGP in most scenarios. Regarding JIC-FCFS and HEFT-FCFS, there is a slight improvement on the total cost between several loose deadlines (i.e., ξ=$12, 18, 24$). Compared to GP-based algorithms, this implies a limitation in their capability to develop better VM rental plans when dealing with loose deadlines.

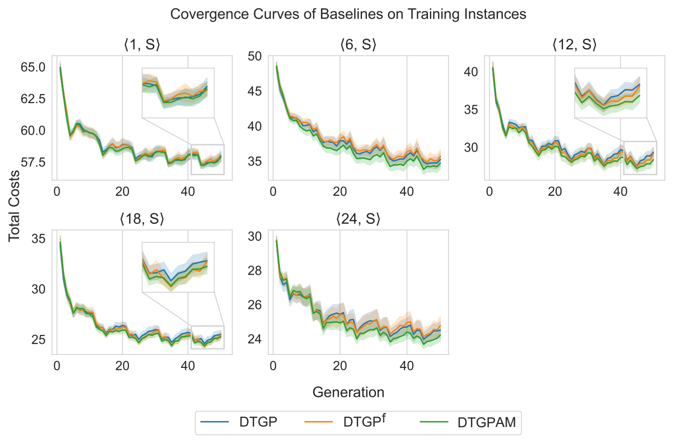


Fig. 8. Convergence curves of the total cost of GP-based algorithms on training instances across 50 independent runs.

Fig. 8 shows the convergence curves of DTGP, DTGP^f and DTGPAM on training instances across 50 independent runs. These convergence curves confirm that all competing GPHH algorithms are quite stable and can converge during the training phase. For several cases, including $\langle 6, S\rangle$, $\langle 12, S\rangle$

TABLE VI
THE MEAN (STANDARD DEVIATION) TOTAL COST OF 5 ALGORITHMS ON 15 TESTING SCENARIOS ACROSS 50 INDEPENDENT RUNS

Scenarios	JIC-FCFS [8], [9]	HEFT-FCFS [6], [9]	DTGP [5]	DTGP^f [5], [31]	DTGPAM
$\langle 1, S\rangle$	63.35(1.06)	61.99(7.35)(+)	56.64(1.32)(+)(+)	56.56(1.01)(+)(+)(≈)	56.43(1.03)(+)(+)(≈)(≈)
$\langle 1, M\rangle$	80.43(1.29)	85.14(1.85)(-)	71.76(3.94)(+)(+)	71.91(3.45)(+)(+)(≈)	71.62(3.60)(+)(+)(≈)(≈)
$\langle 1, L\rangle$	99.45(1.08)	90.61(1.45)(+)	82.47(1.67)(+)(+)	82.38(1.46)(+)(+)(≈)	82.20(1.49)(+)(+)(≈)(≈)
$\langle 6, S\rangle$	73.86(1.31)	39.38(8.42)(+)	32.62(4.32)(+)(+)	32.80(4.34)(+)(+)(≈)	31.97(4.47)(+)(+)(≈)(≈)
$\langle 6, M\rangle$	86.86(0.86)	64.26(1.8)(+)	54.44(7.29)(+)(+)	54.20(6.88)(+)(+)(≈)	52.67(7.01)(+)(+)(≈)(≈)
$\langle 6, L\rangle$	103.82(1.14)	73.1(3.1)(+)	64.43(7.92)(+)(+)	62.47(5.34)(+)(+)(≈)	61.82(5.23)(+)(+)(+)(+)
$\langle 12, S\rangle$	58.17(2.79)	38.11(8.93)(+)	25.67(2.13)(+)(+)	25.31(2.17)(+)(+)(≈)	25.44(2.48)(+)(+)(≈)(≈)
$\langle 12, M\rangle$	79.33(1.65)	62.41(1.97)(+)	45.40(6.54)(+)(+)	45.97(6.41)(+)(+)(≈)	43.81(7.07)(+)(+)(≈)(+)
$\langle 12, L\rangle$	99.54(1.53)	67.2(1.37)(+)	53.46(3.55)(+)(+)	52.54(3.12)(+)(+)(≈)	52.01(3.49)(+)(+)(+)(≈)
$\langle 18, S\rangle$	57.43(2.48)	37.41(9.04)(+)	23.82(1.15)(+)(+)	23.44(0.82)(+)(+)(+)	23.37(0.70)(+)(+)(+)(≈)
$\langle 18, M\rangle$	67.95(1.64)	60.72(1.68)(+)	37.51(3.99)(+)(+)	38.17(4.90)(+)(+)(≈)	37.91(4.12)(+)(+)(≈)(≈)
$\langle 18, L\rangle$	87.67(1.02)	66.27(1.17)(+)	49.79(3.33)(+)(+)	48.54(2.14)(+)(+)(+)	48.85(2.16)(+)(+)(+)(≈)
$\langle 24, S\rangle$	53.34(1.33)	36.88(9.13)(+)	23.03(0.43)(+)(+)	23.33(0.72)(+)(+)(-)	23.08(0.31)(+)(+)(≈)(+)
$\langle 24, M\rangle$	66.15(1.23)	59.59(1.64)(+)	38.62(4.29)(+)(+)	39.04(4.83)(+)(+)(≈)	37.74(3.60)(+)(+)(+)(+)
$\langle 24, L\rangle$	82.46(1.42)	65.15(1.55)(+)	51.13(5.87)(+)(+)	52.28(6.61)(+)(+)(≈)	49.64(4.10)(+)(+)(+)(+)

* $\langle 12, S\rangle$ denotes that $\xi = 12$ and heuristics are tested on the *Mix_Small* scenario.
* (+), (-) or (≈) indicates that the matching result is significantly better, worse, or equivalent to its counterpart.

and $\langle 24, S\rangle$, our proposed DTGPAM appears to converge faster than DTGP and DTGP^f.

In summary, our experimental evidence suggests that DTGPAM is more effective than other competing algorithms for tackling the DCDWSC problem with loose deadlines. Moreover, DTGPAM has been shown to possess an advantage in handling large workflows compared to its counterparts.

B. Parameter Sensitivity Analysis

In order to measure the influence of tree selection-related and terminal selection-related parameters on DTGPAM, parameter sensitivity analysis is performed to find a suitable setting for the initial parameter α_0 in Eq. (18) and the sampling ratio r in Eq. (27). All experiments are trained in the scenario $\langle 12, S\rangle$ with 50 independent runs and then are tested in the scenarios $\langle 12, S\rangle$, $\langle 12, S\rangle$ and $\langle 12, S\rangle$.

For the first parameter α_0, it determines the initial distribution of three probabilities p_{both}, p_{vmsr} and p_{tsr} in $\vec{ts}$. For the second parameter r, it determines the probability of using $\vec{tw}$ to generate the mutated subtree.

1) α_0 for DTGP with tree selection: Table VII shows the mean (standard deviation) total cost of tree selection based DTGP (DTGP^{ts}) with $\alpha_0 \in \{2, 5, 8, 11, 14, 17, 20, 23\}$ on 3 testing scenarios across 50 independent runs. Particularly, $\alpha_0 \in \{2, 8, 23\}$ respectively correspond to the upper bounds of the three values $[p_{both}, p_{vmsr}, p_{tsr}]$ in $\vec{ts}$, namely [0.86, 0.14, 0.0], [0.09, 0.82, 0.10] and [0.0, 0.17, 0.83]. The results suggest that the initial setting of $\vec{ts}$ has negligible impact on the performance of DTGP^{ts}. In other words, DTGP^{ts} has a stable performance under different α_0.

Since DTGP^{ts} is less sensitive to α_0, the value of α_0 can be determined arbitrarily from the range $\{2, 5, 8, 11, 14, 17, 20, 23\}$. According to our preliminary work

TABLE VII
THE MEAN (STANDARD DEVIATION) TOTAL COST OF DIFFERENT α_0 OF DTGP^{ts} ACROSS 50 INDEPENDENT RUNS

α_0	$\langle 12, S\rangle$	$\langle 12, M\rangle$	$\langle 12, L\rangle$
2	30.00(2.01)	45.29(3.15)	63.79(4.62)
5	30.32(2.33)	45.93(3.45)	64.15(4.35)
8	30.00(2.09)	45.33(3.46)	63.74(4.47)
11	29.88(1.96)	45.93(3.32)	64.44(4.25)
14	30.41(2.19)	45.91(3.56)	64.37(4.83)
17	30.01(2.05)	45.48(3.29)	63.65(3.98)
20	29.68(2.05)	45.00(3.51)	63.36(4.87)
23	30.29(1.87)	45.85(2.93)	64.66(4.07)

[5], mutating VMSR often leads to larger changes in individual performance than TSR, hence it is better to concentrate on mutating VMSR in the earlier generations. Consequently, we set $\alpha_0 = 5$ for DTGP^{ts} which corresponds to $\vec{ts}_0 = [0.37, 0.61, 0.02]$.

2) Sampling Ratio r for DTGPAM: Table VIII shows the mean (standard deviation) total cost of DTGPAM achieved with different sampling ratios r and cooling schedule functions on 3 testing scenarios across 50 independent runs. We tested six cooling schedule functions listed in the second column, namely $linear$, log, $expo$, $linear\_rev$, log_rev and $expo\_rev$. These alternative versions, such as $linear$ and $linear\_rev$, are used to verify whether the changing trend of r (rise or fall) had an influence on the results. The results indicate that using a cooling schedule function that gradually increases r (i.e., $linear$, log_rev and $expo\_rev$) yields better performance in comparison to using a cooling

TABLE VIII
THE MEAN (STANDARD DEVIATION) TOTAL COST OF DTGPAM WITH DIFFERENT r ACROSS 50 INDEPENDENT RUNS

Name	$r = f(gen)$	Tendency	$\langle 12, S\rangle$	$\langle 12, M\rangle$	$\langle 12, L\rangle$
$cons$	1	$\longrightarrow$	29.71(1.95)	45.05(3.33)	63.37(4.79)
$linear$	$1 - \frac{50-gen}{50}$	$\nearrow$	30.08(2.08)	45.20(3.20)	63.32(4.50)
log	$\frac{1}{1+2\log(gen+1)}$	$\searrow$	30.06(2.29)	45.74(3.68)	63.99(4.51)
$expo$	0.9^{gen}	$\searrow$	30.10(2.05)	46.23(3.41)	64.61(3.93)
$linear\_rev$	$\frac{50-gen}{50}$	$\searrow$	30.08(2.06)	46.07(4.06)	64.95(5.41)
log_rev	$1 - \frac{1}{1+2\log(gen+1)}$	$\nearrow$	**29.68(2.26)**	**44.77(4.02)**	**62.77(4.68)**
$expo\_rev$	$1 - 0.9^{gen}$	$\nearrow$	29.91(1.99)	45.42(3.81)	63.40(4.50)

* gen denotes the index of generation.

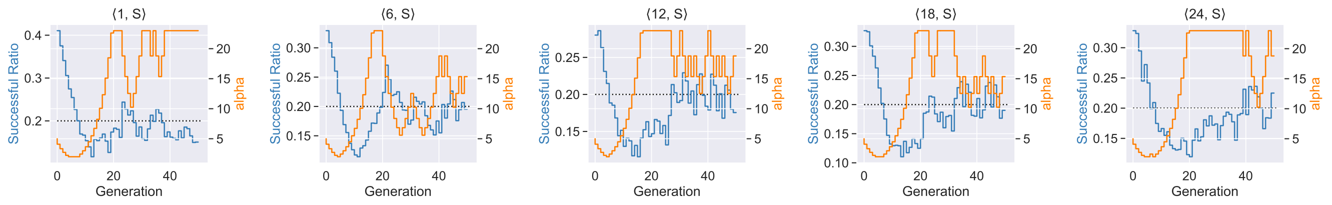


Fig. 9. Curves of $\bar{\phi}$ and α for one run of DTGPAM on 5 training scenarios.

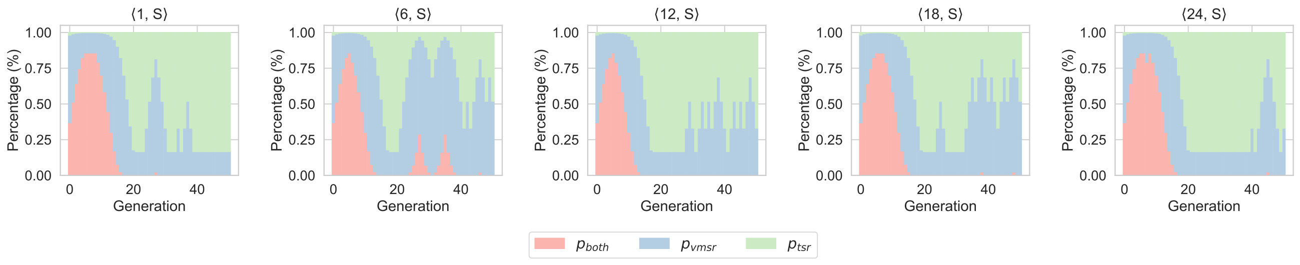


Fig. 10. Distribution of p_{both}, p_{vmsr}, and p_{tsr} for one run of DTGPAM on 5 training scenarios.

schedule function that gradually reduces r (i.e., log, $expo$ and $linear\_rev$).

Further comparing $linear$, log_rev and $expo\_rev$, we found that DTGPAM achieved the lowest average total cost on the three testing scenarios by using log_rev. Due to this observation, we recommend adjusting r during evolution according to log_rev for DTGPAM.

C. Tree Selection Analysis

Fig. 9 shows the variation of $\bar{\phi}$ and α over generations for one run of DTGPAM on 5 training scenarios. We can see that the proposed tree selection mechanism can successfully maintain $\bar{\phi}$ around 1/5 (the gray dotted line) in the later 25 generations. In the early generations of DTGPAM, there is a high successful ratio $\bar{\phi}$ due to the randomness and diversity of the initial population. This is independent of the value of α_0. Referring to Subsection IV-C, if the successful ratio $\bar{\phi}$ is more than 1/5 (corresponding to 0.2 in Fig. 9), we should decrease α. Meanwhile, if $\bar{\phi}$ is less than 1/5, we should increase α.

Fig. 10 shows the distribution of p_{both}, p_{vmsr}, and p_{tsr} across all generations on 5 training scenarios, which corresponds to the same runs in Fig. 9. It can be seen that the variation of $\vec{ts} = [p_{both}, p_{vmsr}, p_{tsr}]$ is consistent with the distribution parameter α in Fig. 9. As α decreases, p_{both} becomes larger. Conversely, as α increases, p_{both} becomes smaller. This realizes adaptive adjustment of $\vec{ts}$ through the 1/5 rule. In addition, p_{both} dominates tree selection in the early generations. Hence mutated individuals tend to undergo large changes in early generations, as we expected.

D. Terminal Analysis

We count the number of terminals among the best 50 rule pairs generated by DTGPAM to identify important terminals. An analysis of their significance in solving the DCDWSC problem was also conducted. The percentage of each terminal type is calculated as the ratio of the number of each terminal type to the total number of terminals in VMSRs or TSRs, formulated by

$$p_i = \frac{1}{50}\sum_{j=1}^{50} \frac{\text{number of } terminal_i \text{ in } tree_j}{\text{total number of terminals in } tree_j} \tag{30}$$

Fig. 11 shows the average percentage of VMSR terminals observed on 50 independent runs for DTGPAM on 5 training scenarios. As deadline constraints become looser (i.e., ξ increases), all terminal types exhibit variations in their relative frequencies. Specifically, certain types (such as ET and NIQ) demonstrate an upward trend, while others (such as LFT) exhibit a decrease. Conversely, certain types (such as TIQ) tend to remain relatively stable. It is noted that the remaining rental period of a VM instance (VMR) plays a crucial role when the deadline constraints are extremely tight (i.e., $\xi = 1$). Specifically, when the execution time of the task exceeds the VMR of the VM instance, VMSRs must determine whether to rent a new VM instance or extend its rental period. As such, VMR has a strong influence on the performance of VMSRs when workflow deadlines are tight.

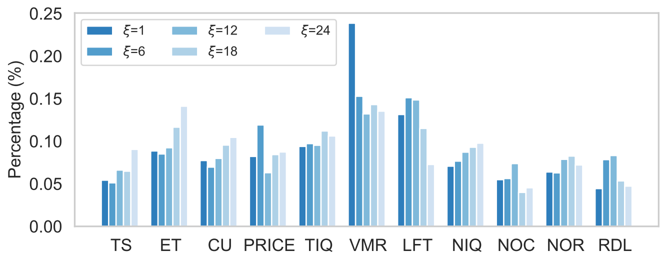


Fig. 11. Terminal statistic for best VMSR of DTGPAM.

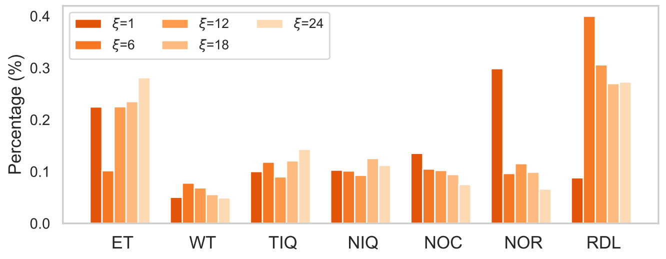


Fig. 12. Terminal statistic for best TSR of DTGPAM.

Fig. 12 shows the average percentage of TSR terminals observed on 50 independent runs for DTGPAM on 5 training scenarios. The top two terminal types appeared in TSRs are ET and RDL, which capture the execution information of the workflow. ET is considered an important terminal type because the execution time of a task (ET) has a significant impact on the waiting time of other tasks in the same VM queue. Meanwhile, the remaining time before the workflow deadline (RDL) also impacts strongly the executing order of those pending tasks in the same VM queue. Besides ET and RDL, our experiment results show that the number of unassigned tasks remaining in a workflow (NOR) has a strong influence on TSRs when workflow deadlines are tight (i.e., $\xi = 1$). This is because NOR captures the current progress toward completing a workflow. It must be considered by TSR to reduce the chance of violating the workflow deadline.

E. Rule Analysis

In order to further understand the evolved scheduling rule, rule analysis is performed on the best VMSR and TSR evolved by DTGPAM in scenario $\langle 24, S\rangle$. These two rules, $\pi_{vmsr}^{(24)}$ and $\pi_{tsr}^{(24)}$, which are simplified by using Mathematica[3], are shown in Eq. (31) and Eq. (32). Particularly, a smaller value calculated by the corresponding rule results in a higher priority for the candidate VM instance or task.

$$\begin{aligned}\pi_{vmsr}^{(24)} = &\min\{\max\{ET, \max\{1, 2ET\} - CU\},\\ &\max\{NOC, TIQ, ET + LFT - VMR\\ &+ \min\{CU, ET\}\}\}\end{aligned} \tag{31}$$

In $\pi_{vmsr}^{(24)}$, the remaining usage time of the VM instance (i.e., VMR) plays an importance role, since its value (> 1000) can be much larger than other terminals ($[10, 500]$). If the priority of candidate VM instances equals to "$\max\{NOC, TIQ, ET + LFT - VMR + \min\{CU, ET\}$" in Eq. (31), then $\pi_{vmsr}^{(24)}$ tends to get smaller values under a larger VMR. That means $\pi_{vmsr}^{(24)}$ prefers to choose VM instances with more available time for the task, rather than those that are nearing expiration. Meanwhile, according to "$\max\{NOC, TIQ\}$". VM instances with shorter waiting queues are more likely to be selected. Meanwhile, if the priority of candidate VM instances equals to "$\max\{ET, \max\{1, 2ET\} - CU\}$" in Eq. (31), then VM instances with faster processing speeds are more likely to be selected.

$$\begin{aligned}\pi_{tsr}^{(24)} = &\max\{ET, ET \cdot RDL \cdot (\min\{NIQ, NOC\} + 1)\\ &+2NOC + \max\{ET, RDL\}\}\end{aligned} \tag{32}$$

In $\pi_{tsr}^{(24)}$, whenever $RDL \leq 0$ (i.e., remaining deadline), the corresponding workflow deadline has been exceeded. When such a rare situation happens under loose deadline constraints (i.e. $\xi = 24$), pending tasks with smaller ET (i.e., shorter processing times) in the VM queue are more likely to be processed first according to Eq. (32). When $RDL > 0$, ET and RDL have significant influence on $\pi_{tsr}^{(24)}$. In this case, pending tasks with smaller ET and smaller RDL (i.e., tighter deadlines) are more likely to be selected. Moreover, $\pi_{tsr}^{(24)}$ prefers to select the pending tasks with a less number of successor tasks.

In brief, the evolved scheduling heuristics/rules are comprised of simple heuristics but are assembled in an effective way that is not easy to be designed manually. Moreover, the interpretability ability of these scheduling heuristics is important and beneficial for practical applications.

F. Successful Ratio Analysis

There are three choices in the tree selection level of adaptive mutation: both rules, VMSR and TSR. We further analyze the successful ratio of these three choices, i.e., $\bar{\phi}_{both}, \bar{\phi}_{vmsr}$ and $\bar{\phi}_{tsr}$, to verify the rationality of how the 1/5 rule is applied in Algorithm 2, defined by

$$\begin{aligned}\phi_i &= \frac{\text{number of successful mutations for using } p_i}{\text{number of total mutations for using } p_i}\\ \bar{\phi}_i &= \frac{1}{MaxGen}\sum_{i=1}^{MaxGen}\phi_i\\ &i \in \{both, vmsr, tsr\}\end{aligned} \tag{33}$$

[3]https://www.wolfram.com/mathematica/

The larger the $\bar{\phi}_{both}, \bar{\phi}_{vmsr}$ or $\bar{\phi}_{tsr}$, the greater its potential to find better offspring.

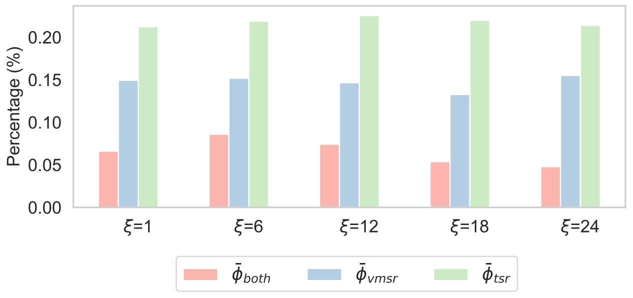


Fig. 13. Average values of $\bar{\phi}_{both}, \bar{\phi}_{vmsr}$ and $\bar{\phi}_{tsr}$ for DTGPAM on 4 training scenarios across 50 independent runs.

Fig. 13 shows the average values of $\bar{\phi}_{both}, \bar{\phi}_{vmsr}$ and $\bar{\phi}_{tsr}$ upon 50 independent runs for DTGPAM on 5 training scenarios. In all scenarios, the ascending order of these three elements is $\bar{\phi}_{both} < \bar{\phi}_{vmsr} < \bar{\phi}_{tsr}$. Given this observation, a larger α in Fig. 7 leads to a larger p_{tsr}, which results in a higher chance of obtaining better mutated offspring. Thus, it is more likely to increase the larger overall successful ratio ϕ_{cur} defined in Eq. (17). For example, $\vec{ts}_1 = [0.05, 0.15, 0.8]$ corresponding to a larger α is more likely to find a large number of high-quality offspring than that of $\vec{ts}_2 = [0.7, 0.2, 0.1]$ corresponding to a smaller α.

The above observation is in line with the principle of the 1/5 rule. As shown in Fig. 9, when $\bar{\phi}$ is less than 1/5, increasing α will result in an increase in p_{tsr} (i.e., the peak of curve moves to the right in Fig. 7), so as to increase $\bar{\phi}$. When $\bar{\phi}$ is more than 1/5, decreasing α will result in an increase in p_{vmsr} (i.e., the peak of curve moves to the left in Fig. 7), so as to decrease $\bar{\phi}$.

G. Terminal Weighting Methods Comparison

An analysis is conducted to compare the performance of frequency-based [31] and MLE-based terminal weighting methods through three scenarios $\langle 24, S\rangle$, $\langle 24, M\rangle$ and $\langle 24, L\rangle$. Following the settings in [31], the utilization of learned terminal weights starts from generation 15. Furthermore, two constant sampling rates $r = 0.2$ and $r - 0.8$ are set to compare their performance under different terminal weight utilizations.

TABLE IX
THE MEAN (STANDARD DEVIATION) TOTAL COST OF TWO TERMINAL WEIGHTING METHODS ACROSS 50 INDEPENDENT RUNS

Scenarios	Frequency-based	MLE-based
$\langle S, 0.2\rangle$	23.28(0.63)	23.22(0.64)(≈)
$\langle M, 0.2\rangle$	38.04(4.6)	38.70(4.92)(≈)
$\langle L, 0.2\rangle$	51.48(6.76)	51.40(7.60)(≈)
$\langle S, 0.8\rangle$	23.37(0.74)	23.19(0.58)(+)
$\langle M, 0.8\rangle$	38.44(4.69)	38.44(4.66)(≈)
$\langle L, 0.8\rangle$	51.13(5.48)	51.38(6.68)(≈)

* $\langle S, 0.2\rangle$ means testing heuristics on scenario *Mix_Small* with $r = 0.2$.

Table IX shows the mean (standard deviation) total cost of frequency-based and MLE-based terminal weighting methods in 6 scenarios across 50 independent runs. It is observed that the MLE-based terminal weighting method is not significantly better than the frequency-based terminal weighting method in all scenarios. The reason may be that the two methods, although distinctly different, in practice only produce slightly different $\vec{tw}$. However we believe the MLE-based method presents a principled way to adjust the weightings of different terminals. It can work reliably well with respect to different sampling ratio r. Therefore, it is the preferred choice in addressing the DCDWSC problem.

VII. CONCLUSIONS

In this paper, we have successfully achieved the overall goal of designing an effective DTGPAM algorithm to jointly evolve effective VMSRs and TSRs for the DCDWSC problem. Three innovative probability vectors were introduced and adaptively updated using the 1/5 rule and the maximum-likelihood technique in DTGPAM. The adapted vectors enable DTGPAM to mutate selected scheduling heuristics in accordance with the impact of each tree and terminal on the performance of the mutated scheduling heuristics.

Experimental results confirmed that the scheduling rules evolved by the proposed DTGPAM algorithm have outperformed several state-of-the-art algorithms on a range of DCDWSC problem instances under multiple different deadline constraints. Through additional experimental analysis, we found that DTGPAM is not sensitive to the initial setting of the tree selection vector. We also performed terminal analysis and identified VMR and LFT as the top two terminals for the VM selection rules. Our analysis further showed that RDL was more frequently used in evolved task selection rules when the deadline constraints became more relaxed. Furthermore, the evolved scheduling heuristics in our experiments were easily interpretable, making them more amenable for practical use.

In the future, we plan to investigate the generalization performance of DTGP on other resource allocation and scheduling problems in the cloud. Furthermore, the mechanism of adaptively updating the probability vectors can be generalized to these GP variants with appropriate modifications to fit the characteristics of each variant. We also plan to explore the potential application of the two proposed mechanisms to other GP variants. Subject to the availability of resources and funding, we also plan to carry out practical evaluation of the scheduling heuristics evolved by DTGPAM in real-world cloud computing applications, in collaboration with our industry partners.

REFERENCES

[1] H. Li, B. Wang, Y. Yuan, M. Zhou, Y. Fan, and Y. Xia, "Scoring and dynamic hierarchy-based nsga-ii for multiobjective workflow scheduling in the cloud," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 2, pp. 982–993, 2022.
[2] V. Arabnejad, K. Bubendorfer, and B. Ng, "Dynamic multi-workflow scheduling: A deadline and cost-aware approach for commercial clouds," *Future Generation Computer Systems*, vol. 100, pp. 98–108, 2019.

[3] V. Huang, C. Wang, H. Ma, G. Chen, and 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.

[4] H. R. Faragardi, M. R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2020.

[5] Y. Yang, G. Chen, H. Ma, and 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.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[7] H. Djigal, J. Feng, J. Lu, and J. Ge, "Ippts: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1057–1071, 2020.

[8] J. Sahni and D. P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2–18, 2015.

[9] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *IEEE International Conference on E-science*. IEEE, 2012, pp. 1–8.

[10] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *IEEE International Symposium on Cluster Computing and the Grid*, vol. 2. IEEE, 2005, pp. 759–767.

[11] M. Xu, Y. Mei, F. Zhang, and M. Zhang, "Genetic programming for dynamic flexible job shop scheduling: evolution with single individuals and ensembles," *IEEE Transactions on Evolutionary Computation*, 2023.

[12] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for online resource allocation in container-based clouds," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020.

[13] Y. Liu, Y. Mei, M. Zhang, and Z. Zhang, "A predictive-reactive approach with genetic programming and cooperative coevolution for the uncertain capacitated arc routing problem," *Evolutionary Computation*, vol. 28, no. 2, pp. 289–316, 2020.

[14] Q.-z. Xiao, J. Zhong, L. Feng, L. Luo, and J. Lv, "A cooperative coevolution hyper-heuristic framework for workflow scheduling problem," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 150–163, 2022.

[15] K.-R. Escott, H. Ma, and G. Chen, "A genetic programming hyper-heuristic approach to design high-level heuristics for dynamic workflow scheduling in cloud," in *IEEE Symposium Series on Computational Intelligence*. IEEE, 2020, pp. 3141–3148.

[16] Y. Yang, G. Chen, H. Ma, M. Zhang, and V. Huang, "Budget and sla aware dynamic workflow scheduling in cloud computing with heterogeneous resources," in *IEEE Congress on Evolutionary Computation*. IEEE, 2021, pp. 2141–2148.

[17] M. Xu, Y. Mei, S. Zhu, B. Zhang, T. Xiang, F. Zhang, and M. Zhang, "Genetic programming for dynamic workflow scheduling in fog computing," *IEEE Transactions on Services Computing*, 2023.

[18] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling," *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1797–1811, 2020.

[19] B. Al-Helali, Q. Chen, B. Xue, and M. Zhang, "Multitree genetic programming with new operators for transfer learning in symbolic regression with incomplete data," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1049–1063, 2021.

[20] A. Friedlander, K. Neshatian, and M. Zhang, "Meta-learning and feature ranking using genetic programming for classification: Variable terminal weighting," in *IEEE Congress of Evolutionary Computation*. IEEE, 2011, pp. 941–948.

[21] I. Rechenberg, *Evolutionsstrategie*. Holzmann-Froboog, Stuttgart, 1973.

[22] R. J. Rossi, *Mathematical statistics: An introduction to likelihood based inference*. John Wiley & Sons, 2018.

[23] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[24] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 472–484.

[25] Y. Bi, B. Xue, and M. Zhang, "Dual-tree genetic programming for few-shot image classification," *IEEE Transactions on Evolutionary Computation*, vol. 26, no. 3, pp. 555–569, 2021.

[26] R. Poli, W. B. Langdon, and N. F. McPhee, *A field guide to genetic programming*. Lulu Enterprises, UK Ltd., 2008.

[27] H. Assimi, A. Jamali, and N. Nariman-Zadeh, "Multi-objective sizing and topology optimization of truss structures using genetic programming based on a new adaptive mutant operator," *Neural Computing and Applications*, vol. 31, no. 10, pp. 5729–5749, 2019.

[28] T. Umeda, Y. Nozaki, and M. Yoshikawa, "Dynamic adaptive mutation based genetic programming for ring oscillator puf," in *International Conference on Computational Intelligence and Applications*, 2018, pp. 210–213.

[29] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.

[30] D. P. Muni, N. R. Pal, and J. Das, "Genetic programming for simultaneous feature selection and classifier design," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 1, pp. 106–117, 2006.

[31] M. Riley, Y. Mei, and M. Zhang, "Improving job shop dispatching rules via terminal weighting and adaptive mutation in genetic programming," in *IEEE Congress on Evolutionary Computation*. IEEE, 2016, pp. 3362–3369.

[32] N. Rizvi, R. Dharavath, L. Wang, and A. Basava, "A workflow scheduling approach with modified fuzzy adaptive genetic algorithm in iaas clouds," *IEEE Transactions on Services Computing*, pp. 1–1, 2022.

[33] C.-H. Youn, M. Chen, and P. Dazzi, *Cloud broker and cloudlet for workflow scheduling*. Springer Singapore, 2017.

[34] B. Hajek, "Cooling schedules for optimal annealing," *Mathematics of Operations Research*, vol. 13, no. 2, pp. 311–329, 1988.

[35] M. Pei, H. An, B. Liu, and C. Wang, "An improved dyna-q algorithm for mobile robot path planning in unknown dynamic environment," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 52, no. 7, pp. 4415–4425, 2021.

[36] A. Jiménez-Martín, A. Mateos, and J. Z. Hernández, "Aluminium parts casting scheduling based on simulated annealing," *Mathematics*, vol. 9, no. 7, p. 741, 2021.

[37] D. Simon, *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[38] A. Slowik and H. Kwasnicka, "Evolutionary algorithms and their applications to engineering problems," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12 363–12 379, 2020.

[39] B. Doerr, C. Doerr, and J. Lengler, "Self-adjusting mutation rates with provably optimal success rules," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 1479–1487.

[40] F. Klinker, "Exponential moving average versus moving exponential average," *Mathematische Semesterberichte*, vol. 58, no. 1, pp. 97–107, 2011.

[41] S. Khakurel, C. Leung, and T. Le-Ngoc, "A generalized water-filling algorithm with linear complexity and finite convergence time," *IEEE Wireless Communications Letters*, vol. 3, no. 2, pp. 225–228, 2014.

[42] S. Wang, Y. Mei, and M. Zhang, "A multi-objective genetic programming algorithm with α dominance and archive for uncertain capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, 2022.

[43] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari, "Online multi-workflow scheduling under uncertain task execution time in iaas clouds," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1180–1194, 2019.

[44] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[45] A. Lissovoi and P. S. Oliveto, *Computational complexity analysis of genetic programming*. Springer International Publishing, 2020.

SUPPLEMENTARY

A. Crossover Comparison

A crossover is defined as successful if the total fitness of the two offspring is better than the total fitness sum of their parents. We applied GPHH approaches with the same parameter setting on the dynamic workflow scheduling (DWS) problem and the symbolic regression (SR) problem. Fig. 14 shows the respective crossover successful ratios. It can be seen that crossover has a lower successful ratio on the DWS problem.

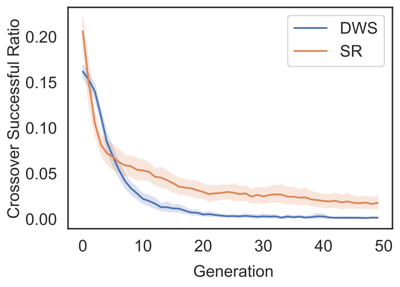


Fig. 14. Crossover successful ratio comparison for DWS and SR problems.

B. Convex Problem

This Supplementary aims to verify Eq. (22) is a convex optimization problem. The convex problem has three additional requirements [23]:

(1) The objective function must be convex.

We set $\varphi = \frac{1-r}{n} \in (0, 1/n]$ in Eq. (22) for ease of derivation, then the objective function is

$$minimum \quad f(p) = -\sum_{i=1}^{n} x_i \cdot \ln(r \cdot p_i + \varphi) \tag{34}$$

The Hessian matrix of Eq. (34) is calculated as

$$\bigtriangledown^2 f = \frac{\partial^2 f}{\partial p} = \begin{bmatrix} \frac{r^2 x_1}{(rp_1+\varphi)^2} & 0 & \cdots & 0 \\ 0 & \frac{r^2 x_2}{(rp_2+\varphi)^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{r^2 x_n}{(rp_n+\varphi)^2} \end{bmatrix} \tag{35}$$

Given $r \in (0, 1], x_i \in [0, +\infty)$ and $\varphi \in (0, 1/n]$, we have $|\bigtriangledown^2 f| \geq 0$, thus the Hessian matrix as shown in Eq. (35) is a positive definite matrix, so that Eq. (34) is a convex function.

(2) The inequality constraint functions as Eq. (22a) **must be convex.**

Since $0 \leq p_i \leq 1$, the constraint functions are line segments, thus Eq. (22a) is obviously convex.

(3) The equality constraint function $h(p) = \sum_{i=1}^{n} p_i - 1$ **must be affine.**

Definition: a set C is *affine* if for any $x_i, x_2 \in C$ and $\theta \in \mathbf{R}$, we have $\theta x_1 + (1-\theta)x_2 \in C$.

Set the set $C = \{p_i | \sum_{i=1}^{n} p_i - 1 = 0\}$, for any two points $p^1 = [p_1^1, p_2^1, \cdots, p_n^1]^T$ and $p^2 = [p_1^2, p_2^2, \cdots, p_n^2]^T$ belonging to C, we have

$$\begin{cases} \sum_{i=1}^{n} p_i^1 - 1 = 0 \\ \sum_{i=1}^{n} p_i^2 - 1 = 0 \end{cases} \tag{36}$$

For any $\theta \in [0, 1]$, we have

$$\begin{cases} \theta(\sum_{i=1}^{n} p_i^1 - 1) = 0 \\ (1-\theta)(\sum_{i=1}^{n} p_i^2 - 1) = 0 \end{cases} \tag{37}$$

$$\overset{\text{add up}}{\Longrightarrow} [\theta \sum_{i=1}^{n} p_i^1 + (1-\theta) \sum_{i=1}^{n} p_i^2] - 1 = 0 \tag{38}$$

Thus, $\theta \sum_{i=1}^{n} p_i^1 + (1-\theta) \sum_{i=1}^{n} p_i^2$ is also in the line $\sum_{i=1}^{n} p_i - 1 = 0$, belonging to the set C. As a result, $C = \{p_i | \sum_{i=1}^{n} p_i - 1 = 0\}$ is *affine*.

In conclusion, Eq. (22) is a convex optimization problem.

C. Validity of Using Water Filling Algorithm

This Supplementary aims to elaborate how to use the water filling algorithm [41] to solve the example shown in Table X.

TABLE X
NUMBER OF EACH TERMINAL TYPE IN THREE TREES

	$terminal_1$	$terminal_2$	$terminal_3$
$tree_1$	1	2	3
$tree_2$	2	2	2
$tree_3$	0	1	3
x_i	3	5	8

Table X gives an example of how to compute $\vec{tw} = [p_1, p_2, p_3]^T$ with respect to three selected trees that use collectively tree terminal types, where $[x_1, x_2, x_3] = [3, 5, 8]$ and $n = 3$. Give the sampling ratio as $r = 1/2$, by Eq. (27), we have

$$\{\frac{3}{\nu^*} - \frac{1}{3}\}^+ + \{\frac{5}{\nu^*} - \frac{1}{3}\}^+ + \{\frac{8}{\nu^*} - \frac{1}{3}\}^+ = 1 \tag{39}$$

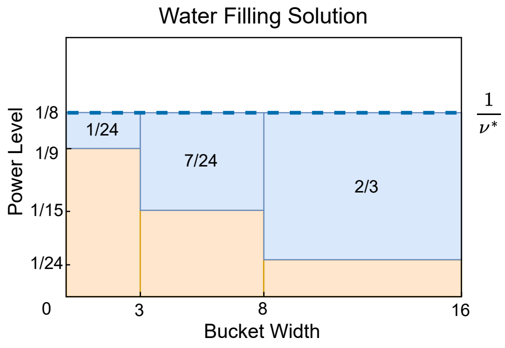


Fig. 15. Example of using the water filling algorithm.

The solution of Eq. (39) is presented in Fig. 15. The width of each bucket is x_i (i.e., 3, 5, 8), and the area of each bucket is $\frac{1-r}{nr} = 1/3$ (i.e., the orange rectangles), thus the height of buckets are 1/9, 1/15, 1/24, respectively. When the water level line $1/\nu^* = 1/8$ is determined, the area from each bucket to the water level line (i.e., the blue rectangles) is the corresponding solution.

$$\begin{cases} p_1^* = (1/8 - 1/9) \cdot 3 = 1/24 \\ p_2^* = (1/8 - 1/15) \cdot 3 = 7/24 \\ p_3^* = (1/8 - 1/24) \cdot 3 = 2/3 \end{cases} \tag{40}$$

To verify the effectiveness of Eq. (27), the optimal p_1^*, p_2^*, p_3^* can also be solved by using the extreme value theorem [37] to maximize Eq. (41). According to Table X, the log-likelihood function is denoted as

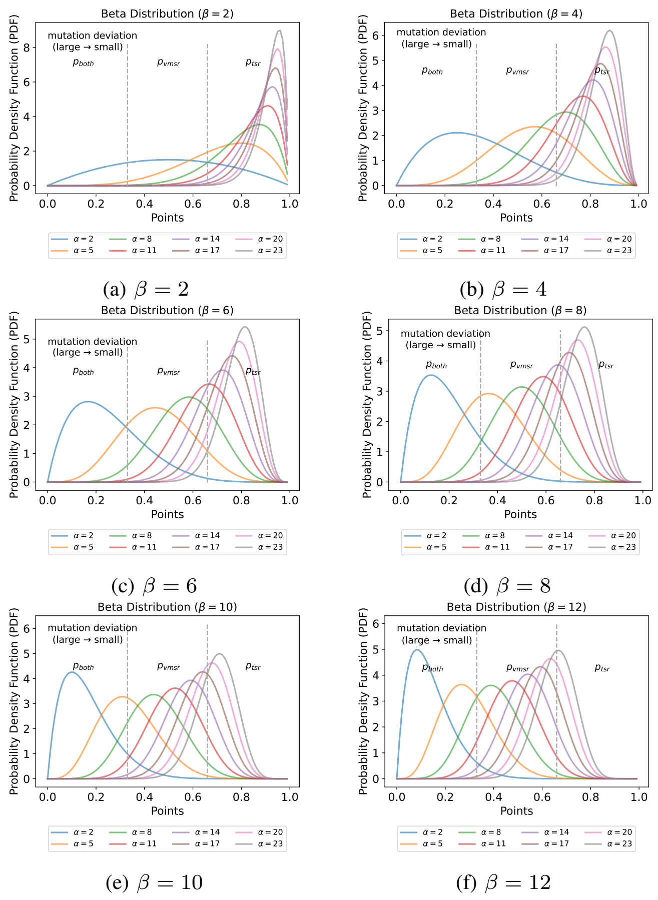
$$\begin{aligned} f(p) = \ln \mathcal{P} &= \ln\left(P(tree_1) \cdot P(tree_2) \cdot P(tree_3)\right) \\ &= 3\ln\left(\frac{1}{6} + \frac{p_1}{2}\right) + 5\ln\left(\frac{1}{6} + \frac{p_2}{2}\right) + \\ &\quad 8\ln\left(\frac{1}{6} + \frac{1 - p_1 - p_2}{2}\right) \end{aligned} \tag{41}$$

Then, Eq. (41) can be solved as follows:

$$\begin{cases} \frac{\partial f}{\partial p_i} = 0 \\ \sum_{i=1}^{3} p_i = 1 \end{cases} \Rightarrow \begin{cases} p_1 = \frac{4-9\cdot p_2}{33} \\ p_2 = \frac{4-5\cdot p_1}{13} \\ p_3 = 1 - p_1 - p_2 \end{cases} \Rightarrow \begin{cases} p_1^* = 1/24 \\ p_2^* = 7/24 \\ p_3^* = 2/3 \end{cases} \tag{42}$$

Clearly, the solution found above matches exactly with the solution obtained by the water-filling algorithm, confirming the correctness of using the water-filling algorithm to compute $\vec{tw}$.

D. Empirical Studies for Beta Distribution



(a) $\beta = 2$ (b) $\beta = 4$ (c) $\beta = 6$ (d) $\beta = 8$ (e) $\beta = 10$ (f) $\beta = 12$

Fig. 16. Distribution map corresponding to different β values.

The parameter $\beta = 8$ is selected based on some numerical studies. For a clear explanation, we presented the results of our numerical studies in Figure 16 of this response letter, which shows the Beta distribution with different β values (i.e., $\beta = 2, 4, 6, 8, 10, 12$). β controls the difference among the three probability values $\vec{ts} = [p_{both}, p_{vmsr}, p_{tsr}]^T$. Moreover, the parameter α is a variable used in the $1/5$ *rule* to adjust the *mutation deviation*, and Figure 16 shows its possible range. When $\beta = 2, 4, 6, 12$, the resulting probability distributions exhibited noticeable imbalances. It can be seen that the slope of the PDF curve for $\beta = 8$ is smaller compared to $\beta = 10$, resulting in relatively small differences among the three probability values obtained from the Beta distribution. In view of the above, we decide to set β to 8 in our experiments.

E. Computational Complexity Analysis

Related notations are introduced in Table XI. We analyze the worst-case complexity of DTGP and DTGPAM using the Big-O notation. The time complexity of DTGP and DTGPAM, denoted by T_{max}^{dtgp} and T_{max}^{dtgpam}, mainly consist of the complexities for evaluation (T_1), parent selection (T_2), crossover (T_3), mutation (T_4) and elitism (T_5). The complexity of each part is listed in Table XII. Specifically, the detailed explanation of each part is as follows:

1) *Fitness evaluation.* In addition to the population size (N) and the number of generations (G), T_1 is also impacted by the complexity for parsing the GP tree, which is $O(d_1 \log d_1)$ following [2]. Thus, the complexity T_1 of fitness evaluation across all generations is $d_1 \log d_1 NG$.
2) *Parent selection.* Given the tournament size as s_k, we have the complexity T_2 of parent selection across all generations as $(s_k + s_k) \times N \times G = 2s_k NG$ [3], where the first s_k is for the random selection of s_k individuals, and the second s_k is for selecting the one with the best fitness among these s_k individuals.
3) *Crossover.* For each crossover operation of two parent individuals, the algorithm executes $2l_1 + 3$ steps, including 1 step to select one of two trees to be mated, $2l_1$ steps to label all nodes in the two trees, and 2 steps to select one crossover point in two trees. Thus, the complexity T_3 of crossover across all generations is $(2l_1 + 3) \times p_c N \times G = (2l_1 + 3)p_c NG$.

TABLE XI
NOMENCLATURE.

Notion	Description	Range
N	population size	1024
G	number of generations	51
d_1	maximum depth of both GP trees, i.e., VMSR and TSR	8
d_2	maximum depth of generated subtrees in mutation	3
l_1	maximum allowed number of nodes in a GP tree subject to the depth limit d_1	2^{d_1}
l_2	maximum allowed number of nodes in a subtree of mutation subject to the depth limit d_2	2^{d_2}
s_1	size of the terminal set used for constructing VMSRs	11
s_2	size of the terminal set used for constructing TSRs	7
p_c	crossover rate	[0,1]
p_m	mutation rate	[0,1]
s_k	the tournament size	7

* an individual consists of a VMSR and a TSR.

TABLE XII
COMPLEXITY ANALYSIS OF DTGP AND DTGPAM.

	Components	DTGP	DTGPAM
T_1	evaluation	$d_1 \log d_1 NG$	$d_1 \log d_1 NG$
T_2	parent selection	$2s_k NG$	$2s_k NG$
T_3	crossover	$(2l_1+3)p_c NG$	$(2l_1+3)p_c NG$
T_4	mutation	$(d_2 \log d_2 + 2)p_m NG$	$(d_2 \log d_2 + 3)p_m NG + +2l_1 NG + (2 + s_1 \log s_1 + s_2 \log s_2)G$
T_5	elitism	NG	NG

4) *Mutation.* In DTGPAM, the complexity T_4 of mutation across all generations consists of three components, i.e., $T_4^{(0)} + T_4^{(1)} + T_4^{(2)}$.
   - Common steps required for mutation ($T_4^{(0)}$): When mutating an individual, the algorithm executes $d_2 \log d_2 + 2$ steps, including 1 step to randomly select one tree, 1 step to randomly select one mutation point in that tree, and $d_2 \log d_2$ steps to generated a subtree. Thus, for $T_4^{(0)}$ we have $(d_2 \log d_2 + 2)p_m NG$, which is bounded by $O(d_2 \log d_2 NG)$.
   - Estimation of tree selection probabilities ($T_4^{(1)}$). In each generation, we need $p_m N$ steps to compare the fitness changes of mutated individuals according to the 1/5 rule, 1 step to calculate the mutation successful ratio, and 1 step to obtain tree selection probabilities by inputting the mutation successful ratio into the Beta distribution. Thus, for $T_4^{(1)}$ we have $(p_m N + 2)G$, which is bounded by $O(NG)$.
   - Estimation of terminal weights ($T_4^{(2)}$). In each generation, we need $2l_1 N$ steps to count the total number of each terminal, and $s_1 \log s_1 + s_2 \log s_2$ steps to calculate the terminal weights of two trees use the water-fill algorithm [4]. Therefore, for $T_4^{(2)}$ we have $(2l_1 N + s_1 \log s_1 + s_2 \log s_2)G$, which is bounded by $O((l_1 N + s_1 \log s_1 + s_2 \log s_2)G)$.

   Finally, the complexity T_4 of mutation in DTGPAM across all generations is $(d_2 \log d_2 + 3)p_m NG + +2l_1 NG + (2 + s_1 \log s_1 + s_2 \log s_2)G$.
5) *Elitism.* In each generation, we need N steps to iterate through the population once to identify the top individuals based on their fitness. Thus, the complexity T_5 of elitism across all generations is NG.

To sum up, the overall complexity of the DTGP and that of our DTGPAM algorithm is calculated by Eq. (43) and Eq. (44), respectively.

$$\begin{aligned} T_{max}^{(dtgp)} =& T_1 + T_2 + T_3 + T_4^{(0)} + T_5 \\ =& [d_1 \log d_1 + 2s_k + (2l_1+3)p_c \\ & \qquad + (d_2 \log d_2 + 2)p_m + 1]NG \\ \leq & c_1(d_1 \log d_1 + s_k + l_1)NG \end{aligned} \tag{43}$$

$$\begin{aligned} T_{max}^{(dtgpam)} =& T_1 + T_2 + T_3 + T_4 + T_5 \\ =& [d_1 \log d_1 + 2s_k + (2l_1+3)p_c \\ & + (d_2 \log d_2 + 2)p_m + 1]NG \\ & + 2l_1 NG + (2 + s_1 \log s_1 + s_2 \log s_2)G \\ \leq & c_2(d_1 \log d_1 + s_k + l_1)NG \end{aligned} \tag{44}$$

where $c_1 > 0$, $c_2 > 0$, $d_1 > d_2$ and $N \geq s_1 log s_1 + s_2 log s_2$. Based on the Big-O method, although the mutation operator of DTGPAM exhibits a higher complexity than that of DTGP, these two algorithms have the same complexity level, i,e, $O((d_1 \log d_1 + s_k + l_1)NG)$.

F. Theoretical Analysis on the MLE-based Method

To make our theoretical analysis feasible, we consider the case where only two terminal types τ_1 and τ_2 exist in GP trees. It is worth noting that the analysis presented here can be extended to cases involving more than two terminal types. Assuming that each GP tree I has n terminal nodes, of which x are terminal type τ_1 and $n-x$ are terminal type τ_2. We further assume that the fitness of the GP tree I depends directly on x and $n-x$, specifically

$$fitness(I) = \begin{cases} 1 & \text{if } x \in \{a, a+1, \cdots, b\} \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

where a and b are integers and $0 < a \leq b \leq n$.

Given any two good GP trees (i.e., $fitness(I_1) = 1$ and $fitness(I_2) = 1$), where the number of terminal type τ_1 in each tree is X_1 and X_2 respectively. For $\forall x_1, x_2$, we have $\mathbb{P}(X_1 = x_1 \cup X_2 = x_2) = \mathbb{P}(X_1 = x_1) \cdot \mathbb{P}(X_2 = x_2)$. In addition, X_1 and X_2 follow a discrete uniform distribution, denoted by

$$\mathbb{P}(X_1 = x_1) = \begin{cases} \frac{1}{b-a+1} & \text{if } x_1 \in \{a, a+1, \cdots, b\} \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

$$\mathbb{P}(X_2 = x_2) = \begin{cases} \frac{1}{b-a+1} & \text{if } x_2 \in \{a, a+1, \cdots, b\} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

According to the calculation steps introduced in our manuscript, we can obtain the terminal weights estimated by the MLE-based [4] and frequency-based [5] methods through Eq. (48) and Eq. (49) respectively.

$$f_{MLE}(x_1, x_2) = \frac{x_1 + x_2}{2n} \tag{48}$$

$$f_{Fre}(x_1, x_2) = \frac{(x_1 + x_2)r}{2n} + \frac{1-r}{2} \tag{49}$$

We denote the weight of terminal type τ_1 as W and the weight of terminal type τ_2 as $1-W$. The distribution of W obtained by these two methods is calculated by

$$\begin{aligned} \mathbb{P}_{MLE}(W = w) = \sum_{x_1=a}^{b} \sum_{x_2=a}^{b} \mathbb{P}(X_1 = x_1) \cdot \mathbb{P}(X_2 = x_2) \\ \cdot \mathbf{1}\{f_{MLE}(x_1, x_2) = w\} \end{aligned} \tag{50}$$

$$\mathbb{P}_{Fre}(W=w) = \sum_{x_1=a}^{b}\sum_{x_2=a}^{b}\mathbb{P}(X_1=x_1)\cdot\mathbb{P}(X_2=x_2) \cdot \mathbf{1}\{f_{Fre}(x_1,x_2)=w\} \tag{51}$$

In our method, a terminal node has $0 \le r \le 1$ probability to be selected according to the terminal weights, and has $1-r$ probability to be randomly selected, as shown in Figure 17. Therefore, the probability of sampling each terminal is calculated by Eq. (52) and Eq. (53).

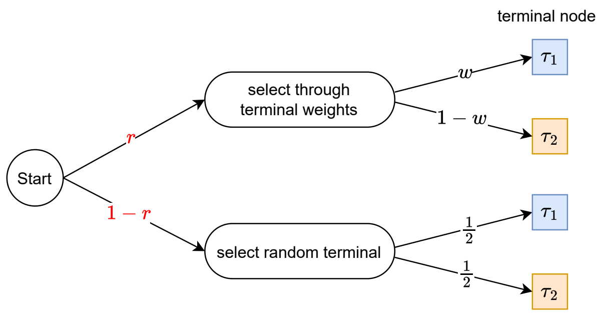


Fig. 17. The mechanism of sampling a terminal node in this article.

$$\mathbb{P}(\tau_1 \mid W=w) = r\cdot w + \cdot\frac{1-r}{2} \tag{52}$$

$$\mathbb{P}(\tau_2 \mid W=w) = r(1-w) + \cdot\frac{1-r}{2} \tag{53}$$

Consequently, the expected fitness of the new individual generated according to the distribution of W in Eq. (50) and Eq. (51) is calculated by

$$EF_{W_{MLE}} = \sum_{w=\frac{a}{n}}^{\frac{b}{n}}\sum_{x=a}^{b}\binom{n}{x}\mathbb{P}_{MLE}(W=w)(rw+\frac{1-r}{2})^x \cdot [r(1-w)+\frac{1-r}{2}]^{n-x} \tag{54}$$

$$EF_{W_{Fre}} = \sum_{w=\frac{ar}{n}+\frac{1-r}{2}}^{\frac{br}{n}+\frac{1-r}{2}}\sum_{x=a}^{b}\binom{n}{x}\mathbb{P}_{Fre}(W=w)(rw+\frac{1-r}{2})^x \cdot [r(1-w)+\frac{1-r}{2}]^{n-x} \tag{55}$$

□

Given the complexity of directly comparing Eq. (54) and Eq. (55), we provide specific values of $EF_{W_{MLE}}$ and $EF_{W_{Fre}}$ under a wide range of different settings of a, b, n and r in Table XIII. It can be observed that $EF_{W_{MLE}}$ is consistently higher than $EF_{W_{Fre}}$ across all parameter settings. This suggests that the MLE-based method effectively addresses the issue of estimating terminal weights.

Compared to the frequency-based method, the MLE-based method leverages mathematical optimization to derive terminal weights that lead to higher expected fitness of newly generated individuals. This translates to a greater likelihood of generating "good" individuals, ultimately improving the overall performance of the DTGPAM algorithm.

TABLE XIII
SPECIFIC VALUES OF $EF_{W_{MLE}}$ AND $EF_{W_{Fre}}$ UNDER SOME PARAMETERS.

a	b	n	r	$EF_{W_{MLE}}$	$EF_{W_{Fre}}$
4	5	8	$\frac{1}{4}$	0.5035	0.4955
4	5	8	$\frac{3}{4}$	0.5110	0.5107
4	5	16	$\frac{1}{4}$	0.1778	0.1124
4	5	16	$\frac{3}{4}$	0.3750	0.3078
4	6	8	$\frac{1}{4}$	0.6497	0.6149
4	6	8	$\frac{3}{4}$	0.6945	0.6870
4	6	16	$\frac{1}{4}$	0.3300	0.2434
4	6	16	$\frac{3}{4}$	0.5321	0.4714
4	7	8	$\frac{1}{4}$	0.7232	0.6572
4	7	8	$\frac{3}{4}$	0.8223	0.8005
4	7	16	$\frac{1}{4}$	0.5039	0.4198
4	7	16	$\frac{3}{4}$	0.6593	0.6190

* $[a, b]$ is the desired range to achieve good fitness, n is the number of terminal nodes in each GP tree, and r is the probability of using terminal weights to sample terminal.