Swarm and Evolutionary Computation 98 (2025) 102113

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Genetically-modified multi-population particle swarm optimization for workflow scheduling in cloud environment

Peiying Zhang [a,b,c], Jingfei Gao [a,b], Lizhuang Tan [c,*], Kai Liu [d,e], Konstantin Igorevich Kostromitin [f,g], Neeraj Kumar [h,i,j]

[a] Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao, 266580, China
[b] Shandong Key Laboratory of Intelligent Oil & Gas Industrial Software, Qingdao, 266580, China
[c] Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, 250014, China
[d] Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing, 100084, China
[e] State Key Laboratory of Space Network and Communications, Tsinghua University, Beijing, 100084, China
[f] Department of Physics of Nanoscale Systems, South Ural State University, Chelyabinsk, 454080, Russia
[g] Institute of Radioelectronics and Information Technologies, Ural Federal University, Yekaterinburg, 620002, Russia
[h] Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala, 147004, India
[i] Department of Computer Science, University of Economics and Human Sciences, Warszawa, 01-043, Poland
[j] Department of Networks and Communications, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam, 31441, Saudi Arabia

ARTICLE INFO

Keywords:
Workflow scheduling
Genetic algorithm
Particle swarm optimization algorithm
Multi-objective optimization

ABSTRACT

As the dominant computing paradigm, cloud computing has become an ideal solution for processing large-scale computing applications, which are often decomposed into massive workflows. However, efficiently scheduling workflow tasks with complex dependencies and allocating appropriate virtual execution units on dynamically fluctuating resources remain significant challenges. Traditional heuristic algorithms often struggle to balance convergence speed and solution diversity when addressing such large-scale multi-objective optimization problems. In this paper, we propose a genetically-modified multi-population particle swarm optimization approach (GMPSO) for workflow scheduling in cloud environments, aiming to achieve a comprehensive optimization of makespan and energy consumption. GMPSO divides the swarm into fitness-based sub-populations and applies distinct genetic operators to each, combining the global search ability of genetic algorithms with the local refinement strength of particle swarm optimization. The (1) load-aware population initialization method, (2) differentiated multi-population search mechanism, and (3) adaptive genetic operator perturbation strategy significantly improve the search quality of GMPSO. We evaluate GMPSO on eight different scales of four types of workflows, and compare its performance with four state-of-the-art algorithms. Experimental results show that GMPSO achieves average improvements of 23.88% in inverted generational distance (IGD), 15.77% in hypervolume (HV) ratio , and 26.01% in spread, demonstrating the excellent overall performance in makespan-energy optimization.

1. Introduction

With the widespread adoption of digital transformation and intelligent application scenarios, the volume of computing tasks has grown exponentially, placing higher demands on computing service capabilities. As a model that delivers computing, storage, and networking services over the Internet, cloud computing provides strong support for large-scale computing tasks [1]. By enabling on-demand subscription mechanisms, it allows enterprises and users to flexibly scale resources, optimize system performance, and reduce operational costs.

In cloud computing environments, computing tasks are often organized through workflow models to improve execution efficiency. A workflow consists of an ordered set of computing tasks [2], each representing an independent computational unit whose execution order is determined by their mutual dependencies. Workflows are typically

* Corresponding author.
E-mail addresses: zhangpeiying@upc.edu.cn (P. Zhang), z23070061@s.upc.edu.cn (J. Gao), tanlzh@sdas.org (L. Tan), liukaiv@tsinghua.edu.cn (K. Liu), kostromitinki@susu.ru (K.I. Kostromitin), neeraj.kumar@thapar.edu (N. Kumar).

https://doi.org/10.1016/j.swevo.2025.102113
Received 13 February 2025; Received in revised form 9 July 2025; Accepted 28 July 2025
Available online 8 August 2025

modeled as directed acyclic graph (DAG) [3], where nodes represent tasks and edges denote the dependencies between the tasks.

Workflow scheduling is a critical technique for managing and optimizing the execution of computational tasks. It aims to ensure that all tasks are completed within the required time constraints while maximizing resource utilization [4]. Specifically, workflow scheduling focuses on two core aspects: determining the execution order of interdependent tasks and assigning them to appropriate virtual execution units. The scheduling strategy is optimized by considering factors such as task priority, type, execution time, and system resources. As a typical NP-hard optimization problem [5], workflow scheduling generally lacks known polynomial-time algorithms for finding exact solutions.

Simple heuristic methods are typically based on greedy strategies, using predefined priority rules to determine the execution order of tasks and the allocation of resources, aiming to quickly generate feasible solutions with low computational overhead. For instance, the Shortest Job First (SJF) algorithm prioritizes tasks with the shortest expected execution times to minimize average waiting time [6]; the Earliest Deadline First (EDF) algorithm schedules tasks according to their deadlines, making it suitable for scenarios with strict real-time requirements [7]; the Minimum Completion Time First (Min–Min) algorithm selects the task–resource pair with the shortest expected completion time among all combinations to improve overall execution efficiency [8]; while the Maximum Completion Time First (Max–Min) algorithm prioritizes the task with the longest earliest completion time, aiming to balance task completion times and prevent long tasks from slowing down overall progress [9]. With the advantage of fast decision-making, these algorithms have achieved satisfactory performance in small-scale workflow scheduling. However, their optimization effectiveness heavily depends on task characteristics and predefined rules, and they lack adaptability to dynamic environments, making them insufficient for handling large-scale and complex workflow scheduling problems.

Metaheuristic algorithms are not dependent on problem-specific rules, but instead explore the solution space through evolutionary and information interaction mechanisms. Examples include: Ant Colony Optimization (ACO), which simulates the behavior of ants constructing paths in the task graph, using pheromones to iteratively improve task sequencing and resource allocation [10]; Genetic Algorithms (GA), which encode task–resource mappings as chromosomes and evolve them through operations such as selection, crossover, and mutation to optimize objectives like makespan or energy consumption [11]; and Particle Swarm Optimization (PSO), which represents scheduling schemes as particles in a multidimensional space, with each particle corresponding to a specific task–resource assignment, the particles update their trajectories based on both personal best and global best solutions [12].

Based on the above discussion, metaheuristic algorithms are indeed effective approaches for workflow scheduling in cloud environments. However, single evolutionary strategies often struggle to balance convergence speed and solution diversity when addressing large-scale multi-objective problems. Specifically, GA perform well in global exploration but typically suffer from poor convergence speed; in contrast, PSO achieves faster convergence but is prone to premature convergence and getting trapped in local optima, which leads to insufficient solution diversity [13]. Motivated by these limitations, we propose a genetically-modified multi-population particle swarm optimization approach that integrates the global search capabilities of GA with the local optimization strengths of PSO, considering both convergence speed and solution diversity.

The contributions of this paper are summarized as follows:

- We propose a genetically-modified multi-population particle swarm optimization approach (GMPSO) to optimize the makespan and energy consumption of workflow scheduling in a cloud environment.
- A load-aware initialization method provides initial solutions with higher availability and quality, significantly improving the search efficiency of GMPSO.
- A differentiated multi-population search mechanism divides the particle population into three sub-populations based on fitness, with each sub-population employing different genetic operators to enrich the search strategies and balance convergence speed and solution diversity.
- An adaptive genetic operator perturbation strategy gradually reduces the influence of genetic operators during evolution, ultimately transitioning to pure particle swarm optimization, resulting in more stable population convergence.
- We evaluate the performance of GMPSO through extensive simulation experiments. The results demonstrate that our method outperforms existing approaches across a range of multi-objective optimization metrics.

The remainder of the paper is organized as follows: Section 2 provides a review of related work. In Section 3, we present the model architecture. Section 4 describes the proposed algorithms based on this model. Simulation results are provided and analyzed in Section 5. Finally, we summarize the paper and outline future work in Section 6.

2. Related work

In this section, we first present an overview of workflow scheduling research in cloud environments, followed by a review of representative algorithms based on two mainstream metaheuristic paradigms: GA and PSO. Furthermore, we summarize recent research on multi-population strategies developed upon these two approaches.

2.1. Workflow scheduling in cloud environment

Workflow scheduling is a critical step in ensuring the efficient execution of large-scale computing applications in cloud environments. As a typical NP-hard problem, it involves multiple constraints, such as task dependencies, resource availability, and execution priorities [14]. With the increasing scale and complexity of computing tasks, workflow scheduling has become a key bottleneck that hinders the improvement of Quality of Service (QoS) in cloud computing. To illustrate this point, several studies have explored workflow scheduling from different QoS objectives. The authors of [15] proposed a dynamic scheduling method based on response latency, which considers task weights and priorities to enhance scheduling efficiency. Meanwhile, the authors of [16] focus on energy consumption, presenting an online scheduling algorithm designed to maximize user fairness and minimize energy consumption. Nonetheless, focusing on a single optimization objective is often insufficient in practical scheduling scenarios.

In order to satisfy multiple QoS demands simultaneously, scholars consider the workflow scheduling problem as a multi-objective optimization problem (MOP) [17]. The authors of [18] proposed a scheduling framework based on deep reinforcement learning, achieving a better trade-off between minimizing delay and energy consumption. Similarly, [19,20] treat energy consumption and disaster tolerance in parallel task systems as a bi-objective combinatorial optimization problem, developing algorithms that exhibit strong overall performance.

Reviewing previous studies, it is evident that makespan and energy consumption are two essential aspects of QoS in cloud workflow scheduling, as they represent the most critical factors affecting system performance. These two metrics also serve as the core optimization objectives of this study. Given the strong optimization potential of metaheuristic algorithms in MOP, the following sections review representative workflow scheduling studies based on two mainstream metaheuristic algorithms.

2.2. Workflow scheduling based on GA

As one of the representative metaheuristic algorithms, GA have been widely applied to workflow scheduling problems in cloud environments. In [11], the authors proposed an adaptive biased random key genetic algorithm for cloud workflow scheduling, which improves scheduling efficiency and achieves better load balancing by refining the integer encoding scheme. The authors of [21] focused on the population selection phase and designed a novel scoring standard based on task completion time and execution cost to select better scheduling individuals, thereby improving scheduling quality under multiple constraints. Nevertheless, during the evolutionary process, the high-quality genes of these well-scored individuals may be lost or disrupted, which can significantly reduce scheduling efficiency. To address this issue, the authors of [22] extracted the longest common subsequence from high-quality chromosomes as elite gene segments and preserved them in the later stages of evolution to maintain genetic stability and improve scheduling efficiency.

However, these methods mainly focus on local improvements to the internal mechanisms of GA and struggle to overcome performance bottlenecks in global search under multi-objective scenarios. To further enhance algorithm performance, some studies have introduced auxiliary mechanisms into GA. In [23], the authors leveraged the rapid convergence and robustness of quantum computing to enhance the quality of GA populations, minimizing execution cost while meeting deadline constraints. The authors of [24] incorporated environment-awareness mechanisms from deep reinforcement learning to more accurately assess the fitness of individuals, adapting to the dynamic scheduling requirements of cloud environments.

Despite the enhancements brought by these auxiliary mechanisms, GA still suffers from limitations in search efficiency and convergence stability. To overcome these limitations, [25] proposed a hybrid optimization method that integrates PSO and GA, achieving a better trade-off between energy consumption and execution cost. This is primarily due to the advantages of PSO in terms of population convergence speed and local refinement. The following section will further review representative studies on workflow scheduling based on PSO.

2.3. Workflow scheduling based on PSO

PSO algorithms have been widely used in recent years due to their excellent performance in scheduling problems. In [12], the authors proposed a specialized bi-level collaborative self-learning PSO algorithm that employs a mixed-integer model to capture the configurations and objectives of the integration problem, thereby maximizing resource utilization. The authors of [26] introduced a stochastic matrix PSO scheduling algorithm, which uses a random integer matrix to represent feasible task scheduling schemes, ultimately achieving the optimal total cost of cloud services. The literature [27] introduces a novel directed non-local convergence PSO algorithm, which significantly reduces both work completion time and execution cost by incorporating nonlinear inertia weights during the directed search process.

Although PSO has excellent search efficiency, population diversity declines in the later stages of the search, and all particles tend to cluster in a smaller region, making it difficult to escape. To address this problem, researchers have explored hybrid optimization strategies that combine PSO with other complementary algorithms. In [28], the workflow scheduling problem was modeled as a constrained optimization problem with a deadline, and a hybrid method alternating between PSO and a critical path approach was proposed, achieving better cost efficiency under time constraints. Meanwhile, the authors of [29] designed a two-phase optimization framework combining PSO with the Whale Optimization Algorithm (WOA). The PSO phase generates promising global solutions used to guide leader selection in WOA, after which WOA further refines the scheduling plan. This integration leverages the strengths of both algorithms, achieving a better trade-off between workflow execution cost and completion efficiency.

These hybrid optimization algorithms have partially alleviated the issues of premature convergence and local optima in PSO, providing valuable insights for our work. However, such sequential integration often suffers from information loss during the alternation of evolutionary strategies, leading to unstable optimization performance, motivating us to pursue a more seamless integration approach. Moreover, due to the homogeneous population design, these algorithms lack diverse evolutionary mechanisms. In this regard, multi-population evolutionary strategies offer a promising alternative. In the following section, we will provide a detailed review of such approaches.

2.4. Workflow scheduling with multi-population strategies

In recent years, multi-population strategies have been widely applied to complex workflow scheduling problems. In [30], a multi-layer particle swarm optimization algorithm was introduced, where particles are divided into several small groups and evolve independently. This design significantly enhances search diversity and reduces the risk of premature convergence. The authors of [31] modeled the scheduling problem as a cost optimization problem under task deadline constraints, employing three distinct bee colonies to co-evolve, embedding global information into the update process of the optimization algorithm, and introducing the Metropolis acceptance criterion to prevent premature convergence of the swarm.

While these methods enhance search diversity through structural multi-population designs, they often lack differentiated guidance tailored to the distinct evolutionary roles or stages of each sub-population. The authors of [32] proposed differentiated knowledge-guided operators tailored to the task sequence features of workflow, enabling adaptive guidance across sub-populations throughout the evolutionary process. In the literature [33], the authors proposed a dynamic multi-population genetic algorithm, with a focus on optimizing makespan and energy, that divides the population into several sub-populations based on fitness. Genetic operators are then designed using the longest common subsequence information among sub-populations, allowing each group to perform differentiated search behaviors that adapt to different evolutionary stages of the scheduling process.

Despite the promising performance of the aforementioned multi-population strategies, two key limitations remain. First, information sharing among sub-populations is often insufficient or delayed. This limited exchange hinders the timely dissemination of high-quality solutions across the entire population, thereby reducing overall convergence efficiency. Second, although several methods introduce differentiated operators to enhance population diversity, such heterogeneity may lead to imbalanced evolutionary dynamics. Specifically, some sub-populations may deviate from the global Pareto direction, compromising the stability of the optimization process in the later stages. These issues motivate us to design a more coordinated and adaptive multi-population framework, as described in the next section.

3. Model of architecture

3.1. Overall architecture model

The overall architecture model of our study is shown in Fig. 1. It mainly consists of three parts: the workflow model, the scheduling execution model, and the cloud resource model. The multi-objective optimization model, which integrates makespan and energy requirements, is embedded in the scheduling model to guide task ordering and resource allocation. The resource module consists of multiple virtual machine (VM) instances of different types, where each VM can execute only one task at a time and waits for tasks in the queue to be processed in sequence. In the resource module, the sequenced tasks are mapped to the corresponding VMs and finally the execution results are returned to the application layer. We will specify the details of the different models in the following section.

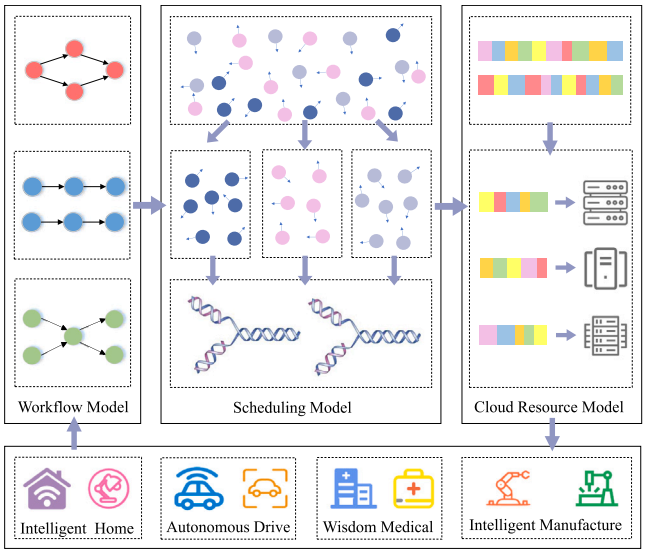
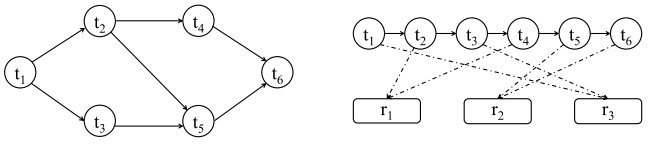


Fig. 1. The overall architecture model of GMPSO.



(a) Workflow DAG containing 6 tasks. (b) Mapping of 6 tasks to 3 VMs.

Fig. 2. Diagram of workflow DAG and task-to-VM mapping.

Table 1
Integer encoding for Task-to-VM mapping.

t	1	2	3	4	5	6
r	3	1	3	1	2	2

3.2. Workflow model

The structure of a workflow is generally abstracted as a DAG, $G = (T, E)$, where $T = \{t_i \mid 1 \le i \le n\}$ denotes a set of tasks; $E = \{e_{ij} \mid 1 \le i \le n, 1 \le j \le n\}$ is the dependency between tasks, D_i represents the size of the data volume of task t_i.

The predecessor and successor sets of task t_i are denoted as $pred(t_i) = \{t_j \mid e_{ji} \in E\}$ and $succ(t_i) = \{t_j \mid e_{ij} \in E\}$, respectively. Task t_i can be executed only after all tasks within $pred(t_i)$ are completed. Similarly, tasks within $succ(t_i)$ cannot be executed until t_i is executed. In a DAG, the task that satisfies condition $pred(t_i) = \emptyset$ is the initial task denoted as t_{entry} and the task that satisfies condition $succ(t_i) = \emptyset$ is known as the end task denoted as t_{exit}. It is worth noting that in a DAG there may be one or more t_{entry} and t_{exit}.

Fig. 2(a) shows a workflow DAG containing 6 tasks, where t_1 is the t_{entry} and t_6 is the t_{exit}. $succ(t_1) = \{t_2, t_3\}$, $pred(t_6) = \{t_4, t_5\}$.

3.3. Cloud resource model

IaaS cloud service providers offer computing services to users by providing different types of VMs. We define a set $R = \{r_1, r_2, \ldots, r_m\}$ to represent the available VMs in IaaS. We use the integer coding table to represent the mapping relationship between tasks and VMs, with the first row representing a set of feasible task sequences in Table 1 and the second row representing the VM number to which the task is assigned.

As shown in Fig. 2(b), the task scheduling sequences are $\{t_1, t_2, t_3, t_4, t_5, t_6\}$, which are mapped to be executed on to VMs $\{r_3, r_1, r_3, r_1, r_2, r_2\}$ respectively. Different types of VMs have different performances, e.g., CPU computation capability and bandwidth affect the task execution time and communication time between resource nodes, respectively. In this paper, we use C_a and B_a to denote the computing capability and bandwidth of VM r_a, respectively. From this, we can derive the execution time of task t_i on VM r_a:

$$T_E(t_i, r_a) = \frac{D_i}{C_a} \tag{3.1}$$

where D_i represents the amount of data for task t_i. The communication time between task t_i and t_j is:

$$T_C(t_i, t_j) = \frac{W_{ij}}{min\{B_a, B_b\}} \tag{3.2}$$

where W_{ij} is the communication data size. The communication bandwidth relies on the VM with the lower bandwidth.

3.4. Scheduling model

3.4.1. Makespan

When calculating the makespan of a workflow task, the constraint relationships between different tasks must be considered. T_S and T_F represent the earliest start execution time and the earliest finish time of a task, respectively, then the earliest start processing time $T_S(t_i, r_a)$ of t_i on r_a is calculated as follows:

$$T_S(t_i, r_a) = max\{T_F(t_j, r_a) + T_C(t_j, t_i)\} \tag{3.3}$$

where t_j is the predecessor task of t_i, i.e., $t_j \in pred(t_i)$, and $T_S(t_i, r_a)$ is the sum of the maximum T_F of all the predecessor tasks of task t_i and the communication time T_C between them. In particular, $T_S = 0$ when t_i is the initial task, i.e., t_{enery}. The earliest completion time of t_i executed in r_a is denoted as $T_F(t_i, r_a)$ and is the sum of $T_S(t_i, r_a)$ and $T_E(t_i, r_a)$ as shown in the following equation:

$$T_F(t_i, r_a) = T_S(t_i, r_a) + T_E(t_i, r_a) \tag{3.4}$$

From the DAG structure mentioned previously, the makespan of workflow sequence is shown as follows:

$$\phi = T_F(t_{exit}, r_a) \tag{3.5}$$

3.4.2. Energy consumption

Generally speaking, the energy consumption generated by the processor accounts for a significant portion of the total energy consumption in the entire IaaS cloud environment system. The processor is classified into running and idle states based on whether tasks are executing on the VMs, with the energy consumption of these states as follows:

$$E_R = \sum_{a=1}^{m} \sum_{i=1}^{n} [U_{i_a} \cdot T_E(t_i, r_a) \cdot Q_{R_a}] \tag{3.6}$$

$$E_I = \sum_{a=1}^{m} \{\phi - \sum_{i=1}^{n} [U_{i_a} * T_E(t_i, r_a)] * Q_{I_a}\} \tag{3.7}$$

where U_{i_a} represents whether the task t_i is running on r_a or not, and has only two values 0 and 1. Q_{R_a} and Q_{I_a} denote the energy consumption during the running and idle states of r_a, respectively. Therefore, the overall energy consumption is expressed as follows:

$$\psi = E_R + E_I \tag{3.8}$$

3.5. Multi-objective optimization model

3.5.1. Optimization objectives and constraints

Based on the above discussion, makespan ϕ and energy consumption ψ are two conflicting optimization objectives because virtual machines with high performance and fast processing capability are usually high energy consuming. This multi-objective optimization problem aims to minimize ϕ and ψ, subject to the following constraints:

$$\begin{aligned} &Minimize\,(\phi, \psi) \\ &\text{s.t.} \quad (C1)\ \sum_{a=1}^{m} U_{i,a} \equiv 1, \qquad i \in \{1, \ldots, n\} \\ &\qquad (C2)\ T_S(t_i) \geq T_F(t_j), \qquad t_j \in pred(t_i) \\ &\qquad (C3)\ T_F(t_i) \leq T_D(t_i), \qquad i \in \{1, \ldots, n\} \\ &\qquad (C4)\ B_{ab} = min\{B_a, B_b\}, \quad a, b \in \{1, \ldots, m\} \end{aligned} \tag{3.9}$$

Here, constraint (C1) ensures that each task is assigned to a unique virtual machine, and (C2) guarantees that tasks are executed in an order consistent with their dependency relationships. Constraint (C3) ensures that no task exceeds its deadline, and (C4) guarantees that the communication bandwidth between tasks stays within the minimum bandwidth available between the assigned virtual machines.

3.5.2. Pareto frontier

In MOP, it is challenging to obtain a solution that is optimal for all objectives. Therefore, the desired solution in such problems is usually a potentially optimal solution for multiple objectives. For any $x_1, x_2 \in X$, if both of the following two conditions are satisfied:

$$\begin{cases} \forall i: \ f_i(x_1) \leq f_i(x_2) \\ \exists j: \ f_j(x_1) < f_j(x_2) \end{cases} \tag{3.10}$$

then x_1 dominates x_2, denoted as $x_1 \prec x_2$. The decision vector x_1 dominates x_2, which implies that none of the optimization objectives of x_1 is worse than x_2, and that x_1 strictly outperforms x_2 in at least one objective. A decision vector x is called Pareto optimal if it is not dominated by any other decision vector in the set. The set of all Pareto optimal solutions in the objective space forms the Pareto frontier (PF).

Fast non-dominated sorting is a key operation proposed in NSGA-II [34] by continuously identifying non-dominated individuals in the population, classifying them into non-dominated ranks, then removing these individuals from the population and continuing to find non-dominated individuals to classify the ranks among the remaining individuals until all of them have been classified.

3.5.3. Crowding distance

Crowding distance is a key metric used in NSGA-II to measure the density of solutions around an individual. It is applied mainly to individuals on the PF with the objective of maintaining the diversity of the population. Suppose τ is an objective function in MOP, and the solutions on the PF are sorted in ascending order according to the value of the objective function $\tau(x)$. Let the sequence of sorted solutions be $I = \{i_1, i_2, \ldots, i_n\}$. Then the congestion distance of solution $i_k (k = 2, 3, \ldots, n-1)$ is calculated as follows:

$$d_{i_k} = \frac{\tau(i_{k+1}) - \tau(i_{k-1})}{\tau_{max} - \tau_{min}} \tag{3.11}$$

where τ_{max} and τ_{min} are the maximum and minimum values of the objective function τ for individuals on the PF, respectively. It is worth noting that the frontier individuals ($k = 1$ *or* n) have neighboring individuals on only one side, so the crowding distance is set to infinity.

The combination of fast non-dominated sorting and crowding distance enables a more precise evaluation of the superiority of individuals. First, the non-dominated sorting ranks of individuals are compared, with individuals having a lower rank performing better overall in multi-objective optimization. For individuals with the same non-dominated rank, their crowding distance is further compared. Individuals with a larger crowding distance help explore a broader solution space, thus maintaining the diversity of the population.

4. Proposed algorithm

4.1. Particle encoding

The PSO algorithm regards each individual as a volume-free particle and the flight trajectory of the population as a continuous optimization process. Particle i is iteratively updated by its position and velocity, the position vector $x_i = [x_{i_1}, x_{i_2}, \ldots, x_{i_D}]$ represents a candidate solution and the velocity vector $v_i = [v_{i_1}, v_{i_2}, \ldots, v_{i_D}]$ is considered as the search direction and step size of the ith particle, where D denotes the dimension of the search space. In the iterative process, the optimization process is based on the following two key mechanisms, i.e., the individual cognition represented by the particle's own historical optimal position p_b and the group cognition represented by the historical optimal position g_b of the whole population. The particles, guided by these two cognitions, are able to utilize the existing information to explore the solution space more efficiently. The update rules for v_i and x_i are defined as follows, respectively:

$$v_i^{t+1} = \omega \cdot v_i^t + c_1 \cdot r_1 \cdot (p_{b_i} - x_i^t) + c_2 \cdot r_2 \cdot (g_b - x_i^t) \tag{4.1}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{4.2}$$

where c_1 and c_2 are two acceleration coefficients, which determine the learning weights of individual cognition and population cognition; r_1 and r_2 are two random numbers generated in the interval [0, 1], which are used to increase the diversity of the particle swarms. ω represents the inertia weight, indicating the ability of the particles to retain previous speeds. A larger inertia weight supports global search, while a smaller one favors local search. To better balance these two search characteristics, we use a linearly decreasing inertia weight as shown below:

$$\omega(n) = \omega_{max} - (\omega_{max} - \omega_{min})\frac{n}{N} \tag{4.3}$$

where n and N represent the current iteration number and the maximum number of iterations, respectively. ω_{max} is the initial inertia weight, and ω_{min} is the inertia weight at the maximum number of iterations.

Let the number of particle dimensions be equal to the total number of workflow tasks, i.e., $D = n$. Then the jth dimension of a particle corresponds to a task t_j of the workflow, and the x_{i_k} value of each dimension corresponds to a VM mapping r_k. Therefore the scheduling problem is transformed into a particle swarm problem and the canonical particle encoding can be represented in Table 1.

4.2. Load-aware initialization

Due to the priority constraints between tasks, the workflow scheduling problem is initialized using a topological sorting method. The set P stores all the tasks that currently have no predecessors or all predecessors have been sorted, and then one of the tasks in P is randomly selected to be added to the sequence T of sorted tasks. This step is repeated until all tasks have been sorted.

Most previous studies have employed a random initialization method to generate the mapping relationship between tasks and VMs. In extreme cases, this approach can result in a large number of sub-optimal individuals, causing premature convergence of the algorithm and making it difficult to escape from local optima. Therefore, a well-designed population initialization is crucial. We will adjust the scheduling scheme according to Table 1, and draw the Gantt chart before and after the modification as shown in Fig. 3, the overall workflow makespan is noticeably prolonged simply by changing the execution unit of task t_1 from r_3 to r_1.

There are two reasons mainly contributing to this result, on the one hand, t_1 has a high priority in the workflow, assigning it to a VM with less processing power takes too long to execute, while the subsequent

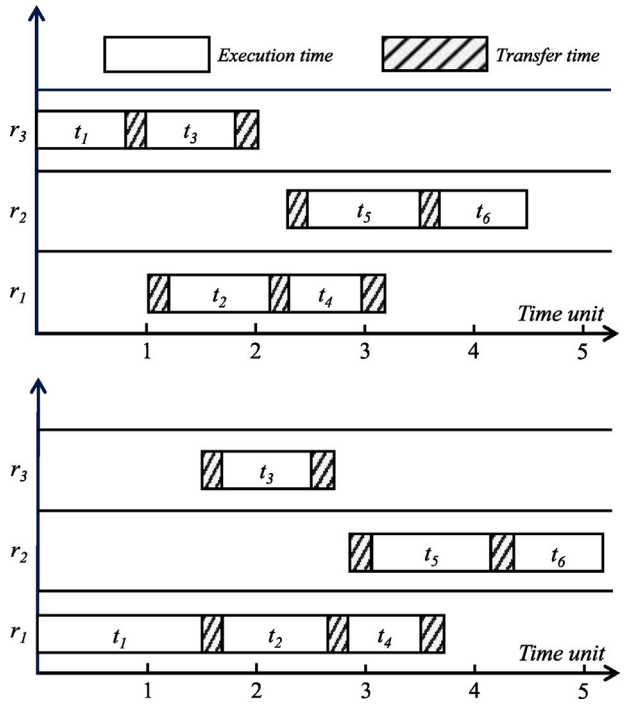


Fig. 3. Gantt charts of scheduling schemes in Table 1.

Algorithm 1 Load-aware initialization

Input: The DAG of workflow, the VM pool;
Output: The scheduling solution $S = (T, R)$;
1: let G be the set of all tasks in the DAG;
2: $T, R = \emptyset$;
3: **while** $G \neq \emptyset$ **do**
4: $P = \emptyset$;
5: **for** $t_i \in G$ **do**
6: **if** $pred(t_i) \subseteq Y$ **then**
7: $P = P + t_i$;
8: **end if**
9: **end for**
10: pick a random task t_j in P;
11: $G = G - t_j, T = T + t_k$;
12: **end while**
13: group VMs by processing capacity A, B;
14: calculate the average volume avg of tasks;
15: **for** $t_i \in T$ **do**
16: **if** $D_i \geq avg$ **then**
17: $R_{index} = min\{T_F(A)\}$;
18: **else**
19: $R_{index} = min\{T_F(B)\}$;
20: **end if**
21: **end for**
22: **return** $S = (T, R)$;

tasks have to wait. On the other hand, r_3 VM is assigned too few tasks, which causes it to be idle for a long time, wasting computational resources.

Based on the above discussion, we fully consider the heterogeneity between tasks and VMs and propose a load-aware initialization scheme. Tasks with data volumes greater than and less than the average are assigned to the high-performance VM group A and the low-performance group B, respectively, and then tasks are prioritized for assignment to the VM with the lightest workload within each group.

Algorithm 2 Crossover based on topological ordering

Input: Parent1, Parent2, P_c;
Output: Offspring1, Offspring2;
1: TrimParent1 = [], TrimParent2 = [];
2: c = Random(1, n);
3: p = Random(0, 1);
4: **if** $p \geq P_c$ **then**
5: **return** Parent1, Parent2;
6: **end if**
7: Segment1 = Parent1[1:c];
8: Segment2 = Parent2[1:c];
9: **for** gene in Parent1 **do**
10: **if** gene not in Segment2 **then**
11: Add gene to TrimParent1;
12: **end if**
13: **end for**
14: **for** gene in Parent2 **do**
15: **if** gene not in Segment1 **then**
16: Add gene to TrimParent2;
17: **end if**
18: **end for**
19: Offspring1 = Concat(Segment1, TrimParent2);
20: Offspring2 = Concat(Segment2, TrimParent1);
21: **return** Offspring1, Offspring2;

Algorithm 1 outlines the detailed process of population initialization. If all predecessor tasks $pred(t_i)$ of a task t_i are already in T, the task t_i is stored in P (lines 5–9). Later on, a task t_j is randomly selected from P to be added to T and G is updated (lines 10–11). When the list of tasks in G is empty, the task sequence initialization is complete. In order to initialize the task and VM mapping sequence R, the average data volume of the tasks should be calculated first and the VMs should be grouped by processing power (lines 13–14). Depending on whether the data volume is greater or less than avg, the task is assigned to the VM with the lightest workload in group A or B (lines 15–21). The initialization sequence for task and VM mapping is finally obtained (line 22).

4.3. Genetic operator

4.3.1. Crossover

The crossover operation in GA produces new offspring by exchanging and combining gene fragments from good individuals. The offspring can not only inherit the excellent genes of the parent individuals, but also enrich the gene pool of the population. In the scheduling problem, the crossover operation is subdivided into task sequence crossover and mapping sequence crossover due to the integer coding specificity of Table 1.

Due to the sequential constraints between tasks, their relative positions cannot be changed when the task sequences are crossovered, we use the crossover method based on topological ordering, and Algorithm 2 gives the concrete steps of implementation. First, randomly select the crossover point on the chromosome (line 2), determine whether to perform the crossover operation by the crossover probability P_c (lines 3–6), the region between the initial task and the crossover point is the crossover segment (lines 7–8), and the new individual is generated by splicing it to the head of another chromosome (lines 19–21) that removes duplicate genes (lines 9–18). Fig. 4 shows the detailed process of task sequence crossover.

For mapping sequence crossover, constraints are not considered, as each task can be assigned to any VM, as illustrated in Fig. 5. The crossover operation can be accomplished by simply swapping the crossover segments.

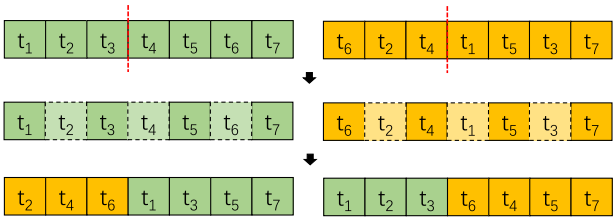


Fig. 4. Task sequence crossover operation.

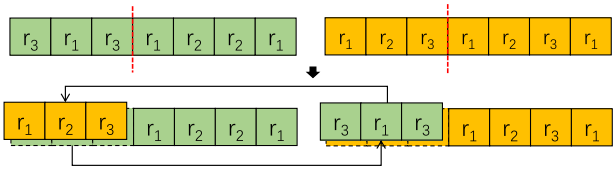


Fig. 5. Mapping sequence crossover operation.

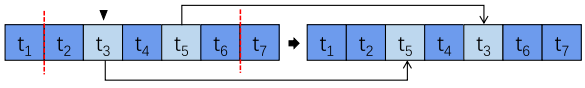


Fig. 6. Task sequence mutation operation.

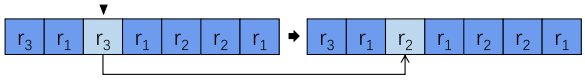


Fig. 7. Mapping sequence mutation operation.

4.3.2. Mutation

As the GA progresses iteratively, individuals in the population may gradually converge. Mutation introduces random changes in the genes of individuals, generating new gene combinations to explore different characteristics, which increases the diversity of the population and helps escape local optimal solutions. Mutation and crossover operations work together to drive the evolution of the population. Similar to the crossover operation, the mutation operation is also divided into task sequence mutation and mapping sequence mutation.

The genes cannot be mutated arbitrarily due to the dependency between tasks. As shown in Fig. 6, we use the task-constrained local exchange strategy for the mutation operation, and Algorithm 3 also gives the procedure for detailed implementation. First confirm the mutation point t_i (line 1), determine whether to perform the mutation operation by the mutation chance P_m (lines 2–5), then traverse the chromosome genes in the forward order, stop when all the predecessors of t_i are retrieved, and record the stopping position t_a (lines 6–10); at the same time, traverse the chromosome in the reverse order, stop when all the successors of t_i are retrieved, and record the stopping position t_b (lines 11–15). At this point t_i can be interchanged with any gene in the $[t_a, t_b]$ region to complete the mutation operation (lines 16–18).

Similar to mapping sequence mutation, mapping sequence mutation takes no constraints into account, as shown in Fig. 7. The mutation is performed by randomly changing the value at the point of mutation to another VM number.

4.4. Differentiated multi-population framework

Our research is based on a genetically modified multi-population PSO framework. According to the evaluation results of individual fitness values, all individuals in the population are classified into superior, medium, and inferior subpopulations, with each subpopulation adopting specific particle evolution parameters and genetic operators for modification.

Algorithm 3 Mutation based on local exchange

Input: Chromosome, P_m;
Output: Mutated-Chromosome;
1: i = Random(1, n);
2: p = Random(0, 1);
3: **if** $p > P_c$ **then**
4: **return** Chromosome;
5: **end if**
6: **for** a=1 to n **do**
7: **if** $pred(t_i) \subseteq \{t_1, ..., t_a\}$ **then**
8: break;
9: **end if**
10: **end for**
11: **for** b=n to 1 **do**
12: **if** $succ(t_i) \subseteq \{t_b, ..., t_n\}$ **then**
13: break;
14: **end if**
15: **end for**
16: m = Random(a,b);
17: swap(t_i ,t_m);
18: **return** Mutated-Chromosome;

Concentrating on optimizing high-quality solutions, the superior population incorporates a small probability crossover operator for perturbation based on particle swarm operation. Through genetic recombination among excellent individuals, it not only generates more promising offspring but also prevents the population from prematurely converging to a local optimum. On the contrary, the inferior population focuses more on the diversity of the population and fully exploits the whole solution space, introducing a mutation operator with higher probability on the basis of the particle swarm, expanding the search range to provide more potential directions for the superior and medium populations. The medium population uses both the crossover operator and the mutation operator to modify the evolution of the particle population, which serves as a link between global search and local convergence, balancing the exploitation and exploration of the population.

As the population evolves and the solution gradually converges to the PF, stability becomes increasingly critical. We control the probability of crossover and mutation by a sigmoid function, gradually weakening the perturbation of the genetic operator on the particle population with the following formula:

$$P(n) = P_G \cdot \frac{1}{1 + e^{\lambda(\frac{n}{N} - 0.5)}} \tag{4.4}$$

where P_G stands for the initial probability of the genetic operator, n and N represent the current number of iterations and the maximum number of iterations, respectively, λ is a parameter that controls the speed of change of the function. As the number of iterations increases, the influence of genetic operators gradually diminishes, eventually transitioning into a pure particle swarm optimization process.

4.5. GMPSO

Algorithm 4 gives the exact steps of GMPSO. First an external archive is defined to hold the set of non-dominated solutions during the iteration process, and then initialize the population (lines 1–2). Then the non-dominated sorting is performed by the objective function, updating the external archive and dividing the population (lines 4–6). Update p_b and obtain g_b by congestion ordering in the external archive, then perform particle position and velocity updates (lines 6–8). Different genetic operator modifications are added on different populations and the chance parameter is used to decide whether to perform a genetic operation or not (lines 9–14). The above steps are repeated until the number of iterations is reached, and finally the external archive is filtered and the non-dominated solution is output (lines 16–17).

Algorithm 4 GMPSO

Input: DAG, the VM pool, maximum iterations N;
Output: Non-dominated solution;

```
Define Archive, Superior, inferior, medium;
Initialize the population pop (Algo. 1);
while i ≤ N do
    Non-dominated sort pop;
    Update the Archive;
    if j < 1/3 * n then
        add pop[j] to Superior;
    else if j > 2/3 * n then
        add pop[j] to Inferior;
    else
        add pop[j] to Medium;
    end if
    Update p_b and g_b;
    for k in each subpop do
        Update v_k and x_k;
        Crossover based on mapping or tasks(Algo. 2);
        Mutation based on mapping or tasks(Algo. 3);
    end for
end while
Filtering the Archive;
return Non-dominated solution;
```

4.6. Complexity analysis

The computational complexity of our proposed GMPSO is related to the number of tasks n, the population size p, and the number of population iterations N. First, for population initialization, the complexity of the task priority sorting part is $O(n^2)$, and the algorithm complexity of the virtual machine mapping stage is $O(n)$. Therefore, the total complexity of population initialization is $O(n^2)$.

In each population iteration, non-dominated sorting usually requires pairwise comparison of individuals in the population, with a complexity of $O(n^2)$. The complexity of sub-population division and particle update is $O(p)$, and the operations of particle crossover and mutation operators both have a complexity of $O(pn^2)$. So the time complexity of the population iteration stage is $O(Nn^2 + Np + Npn^2)$.

Based on the above analysis, the total complexity of GMPSO is $O(n^2 + Nn^2 + Np + Npn^2)$. Considering only the leading term of the highest order, the complexity is $O(Npn^2)$.

5. Performance evaluation

5.1. Experimental settings

To evaluate the proposed approach, we conducted extensive simulation experiments employing WorkflowSim [35] as the simulation framework. This framework is developed based on CloudSim [36] and is specifically optimized for workflow scheduling problems in scientific computing. The workflow dataset employs a series of benchmark workflows, which all simulate real-world scientific applications with different characteristics [37]. CyberShake is proposed by the Southern California Earthquake Center to characterize earthquake hazards, and Epigenomics is a workflow dedicated to processing and analyzing epigenetic data in genomics. The Inspiral workflow is commonly used to simulate and analyze astrophysical processes in gravitational wave events. The Montage workflow is mainly used for the processing of massive astronomical images. The DAG structure for the above four workflows is shown in Fig. 8.

As shown in Table 2, we employ the scheme based on Amazon EC2 as the cloud resource configuration for the experiment. We used six different types of virtual machines. MFLOPS represents millions of floating-point operations per second of the processor and is employed to evaluate the computational capabilities of virtual machines. Q_R and Q_I represent the energy consumption during runtime and idle time, respectively.

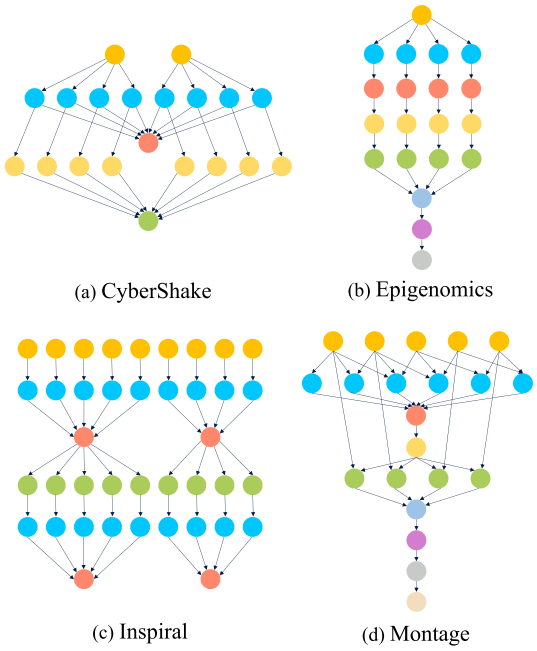


Fig. 8. Four simulation workflow DAGs of Pegasus project.

Table 2
Instance types based on Amazon EC2.

Type	MFLOPS	Memory (GB)	Q_R(w)	Q_I(w)
m1.small	4400	2	0.5	0.1
m1.medium	8800	4	1	0.1
m1.large	17 600	8	2	0.2
m1.xlarge	35 200	12	4	0.2
m3.xlarge	57 200	24	6	0.4
m3.2xlarge	114 400	48	10	0.4

The experiments are divided into three parts. First, orthogonal experiment design (OED) is conducted to tune the parameters. Second, an ablation study is performed to analyze the effectiveness of different modules within the algorithm. Finally, four popular algorithms are selected for comparison to evaluate the comprehensive performance in multi-objective optimization.

5.2. Parameter tuning

As shown in the above analysis, the population size p and the number of evolutionary iterations N are the key factors affecting computational complexity. We conduct parameter tuning for these two variables based on OED. Each parameter is set to three levels: $p \in \{40, 50, 60\}$ and $N \in \{100, 200, 300\}$. Experiments are conducted on three simulated workflows of different scales. The results are presented in Table 3, where C denotes different combinations of parameter settings, and F represents the average value at each level of a single parameter.

The experimental results in Table 3 indicate that C_2 represents an ideal parameter combination. This is attributed to the fact that an insufficient population size often fails to produce a diverse and promising initial solution set, thereby prolonging the exploration phase during the evolutionary process. Conversely, an excessively large population tends to slow down the convergence of the algorithm. Similarly, if the number of iterations is set too low, the population may be

Table 3
Orthogonal experimental design for parameter tuning.

C	p	N	Ep-24	In-100	Mo-1000
C_1	1(40)	1(100)	1.93e+3	3.14e+3	2.30e+3
C_2	1(40)	2(200)	1.95e+3	3.29e+3	2.31e+3
C_3	1(40)	3(300)	2.02e+3	3.35e+3	2.39e+3
C_4	2(50)	1(100)	1.98e+3	3.24e+3	2.33e+3
C_5	2(50)	2(200)	1.84e+3	2.95e+3	2.25e+3
C_6	2(50)	3(300)	1.89e+3	3.24e+3	2.46e+3
C_7	3(60)	1(100)	1.93e+3	3.27e+3	2.50e+3
C_8	3(60)	2(200)	1.97e+3	3.34e+3	2.56e+3
C_9	3(60)	3(300)	2.09e+3	3.59e+3	2.79e+3
F_1	2.52e+3	2.51e+3	–	–	–
F_2	**2.46e+3**	**2.49e+3**	–	–	–
F_3	2.67e+3	2.67e+3	–	–	–
R	p_2(50)	N_2(200)	–	–	–

terminated before reaching maturity, making it difficult to produce competitive individuals. In contrast, an overly large number of iterations can lead to redundant computations and waste of resources. Based on the above analysis, the population size p and the number of evolutionary iterations N are set to 50 and 200, respectively.

The additional experimental parameters are detailed in Table 4 below. We assign a small crossover probability to the superior subpopulation to encourage the combination of elite genes. In contrast, a higher mutation probability is set for the inferior population to enhance the exploration of potentially good solutions in the search space. The genetic and particle swarm parameters for the medium population are set using classical values that are generally effective for most problems.

5.3. Component validity analysis

5.3.1. Initialization method

Our study proposes a load-aware population initialization method. To verify the effectiveness of this approach, we conducted experiments comparing it with random initialization on simulation workflows of varying scales. The average performance on a logarithmic scale is presented in Fig. 9.

Fig. 9(a) shows the workflow execution times obtained from the experiments. It is evident that our method achieves higher optimization efficiency, especially on large-scale and complex workflows. This can be attributed to the fact that our initialization approach provides favorable initial conditions for the optimization process, effectively reducing task waiting times caused by suboptimal virtual machine mappings.

Furthermore, we examined the impact of the two initialization methods on load balancing. Specifically, we used the following formula to evaluate the load balance among virtual machines:

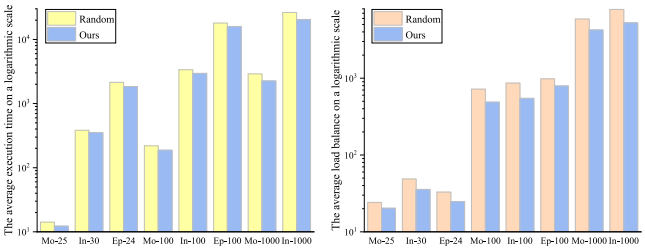
$$LB = \sum_{j=1}^{m} (L_j - \bar{L})^2 \tag{5.1}$$

where L_j denotes the number of tasks assigned to virtual machine r_j, and $\bar{L}$ represents the average number of tasks across all virtual machines. As illustrated in Fig. 9(b), our method achieves better load balancing compared to the random population initialization approach, which is essential for improving overall system performance and resource utilization.

5.3.2. Evolutionary strategy

Our study introduces an enhanced multi-population evolutionary strategy with optimized genetic operators. To verify the effectiveness of this strategy, it is compared with a traditional baseline—the standard PSO without any modifications. Following that, Fig. 10 presents the comparison results of the two strategies on four different simulated workflows.

It is evident that the solutions obtained by our strategy are closer to the Pareto front, and their dominance over those produced by PSO



(a) Execution time comparison. (b) Load balance comparison.

Fig. 9. Log-Scale performance comparison of two initialization methods.

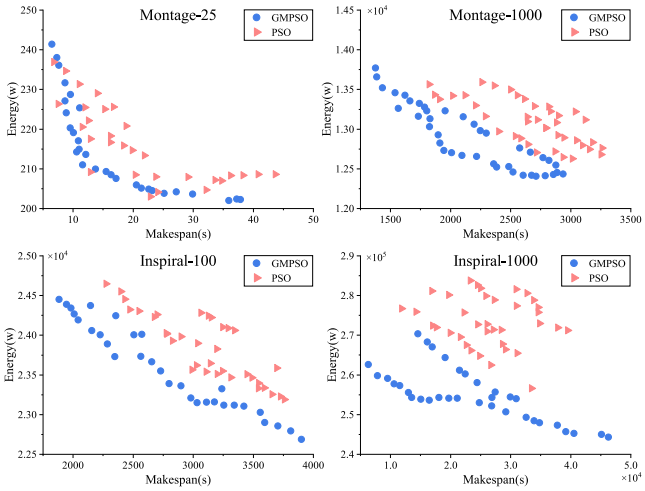


Fig. 10. Performance comparison of two evolutionary strategies.

becomes increasingly pronounced as the workflow scale grows. We also observe that the solutions generated by PSO are scattered, while our approach achieves better convergence. This is attributed to the guiding role of the superior subpopulations, which enable faster identification of high-quality regions. Notably, for the two large-scale workflows, the PSO solutions are concentrated in a local region, whereas our method discovers promising solutions across multiple regions. This leads to a preliminary conclusion that our multi-population strategy demonstrates a broader exploration capability in the solution space.

5.4. Comparison with multi-objective algorithms

5.4.1. Benchmark algorithms

In this section, we select four benchmark algorithms that are widely recognized in the field, namely DMGA [33], MHPSO [30], HPSO [38] and MOH [39]. HPSO is a multi-objective particle swarm optimization algorithm based on non-dominated sorting, while MOH is an improved algorithm developed based on NSGA-II. DMGA and MHPSO are the most representative multi-population algorithms based on genetic algorithms and particle swarm optimization, respectively. All parameter settings for the baseline algorithms follow recommendations from the literature and are detailed in Table 4.

5.4.2. Evaluation metrics

(1) *Inverted Generational Distance (IGD)*: IGD is an evaluation metric for comprehensive performance in MOP. It primarily assesses the convergence and distribution performance by calculating the minimum sum of distances between each point on the PF and the set of individuals obtained by the algorithm. A lower IGD value indicates better

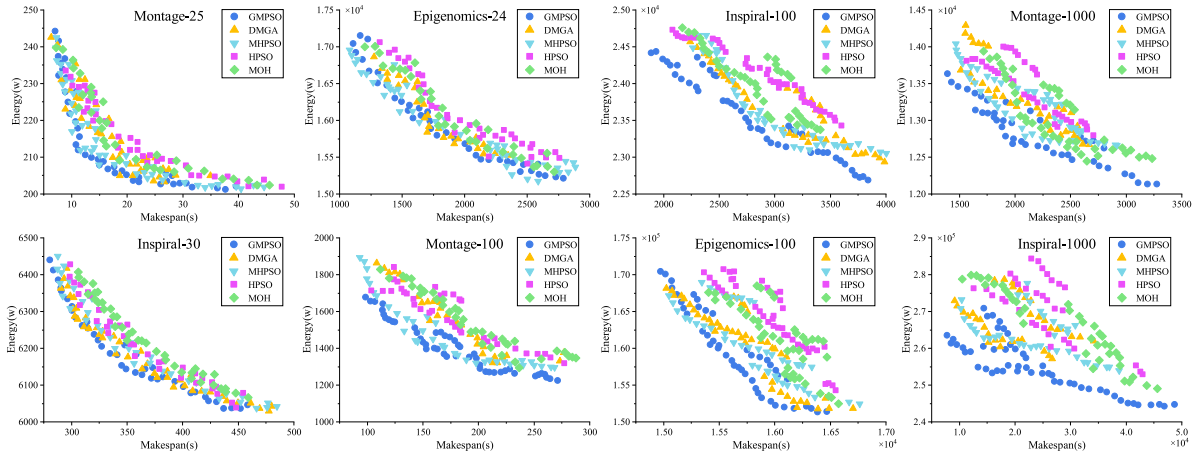


Fig. 11. Makespan-Energy optimization results of five algorithms on different workflows.

Table 4
Parameter settings of all benchmark algorithms.

Algorithm	Parameter
GMPSO	$\omega_{max} = 0.9, \omega_{min} = 0.4, c_1 = c_2 = 2, \lambda = 10$ $p_{cs} = 0.2, p_{cm} = 0.6, p_{mm} = 0.05, p_{mi} = 0.1$
DMGA	$ps = 100, k = 10, p_c = 1, p_m = 1/n$
MHPSO	$g_{max} = 15, \mu = 2.0, \theta_{em} = 0.75$
HPSO	$\omega = 0.9 \sim 0.1, c_1 = 2.5 \sim 0.1, c_2 = 0.5 \sim 2.5$
MOH	$p_c = 1, p_m = 1/n, k = 1.5 \sim 4.0$

overall performance, including both convergence and distribution. The formula for IGD is given below:

$$IGD = \frac{\sum_{x \in P^*} \min_{y \in P} dist(x, y)}{|P^*|} \tag{5.2}$$

where P^* denotes a set of uniformly distributed reference points on the true Pareto front, P represents the solution set obtained by the algorithm, and $dist(x, y)$ is the Euclidean distance from a point $x \in P^*$ to its nearest neighbor $y \in P$.

(2) *Hypervolume (HV)*: HV is a metric commonly used in MOP to evaluate the coverage and uniformity of the distribution of the solutions. HV quantifies the performance of the solution set by selecting a single reference point and calculating the volume of the hypercube formed by all the non-dominated solutions in the set with respect to the reference point. Obviously, a higher value of HV implies a better quality and diversity of the resulting solution set. The formula for HV is as follows:

$$HV = \delta\left(\bigcup_{i=1}^{|S|} v_i\right) \tag{5.3}$$

where δ denotes the Lebesgue measure, $|S|$ represents the total number of non-dominated solutions, and v_i denotes the hypervolume formed by the ith solution and the reference point.

(3) *Spread*(Δ): Δ is a metric used in MOP to evaluate the distribution of the solution set. It mainly assesses the uniformity of the solutions obtained in the objective space and the adequacy of their coverage of the PF. A smaller Δ value indicates a more uniform distribution of the solutions and broader coverage of the objective space. The Δ is calculated as follows:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}} \tag{5.4}$$

here, d_f and d_l represent the Euclidean distances between the extreme solutions and the obtained boundary solutions, d_i denotes the Euclidean distance between consecutive solutions, and $\bar{d}$ is the average of d_i.

Table 5
Wilcoxon test on three metrics between algorithms.

		GMPSO	DMGA	MHPSO	HPSO	MOH
GMPSO	IGD	–	2.07e−3	4.97e−3	1.82e−3	1.82e−3
	HV_r	–	4.91e−2	9.05e−3	1.81e−3	1.81e−3
	Δ	–	2.07e−3	4.97e−3	1.80e−3	1.82e−3
DMGA	IGD	2.07e−3	–	1.63e−2	1.82e−3	1.80e−3
	HV	4.91e−2	–	1.98e−1	1.81e−3	1.82e−3
	Δ	2.07e−3	–	1.63e−2	1.82e−3	1.81e−3
MHPSO	IGD	4.97e−3	1.63e−2	–	1.82e−3	1.82e−3
	HV_r	9.05e−3	1.98e−1	–	1.82e−3	2.07e−3
	Δ	4.97e−3	1.63e−2	–	1.80e−3	1.80e−3
HPSO	IGD	1.82e−3	1.82e−3	1.82e−3	–	2.78e−1
	HV_r	1.81e−3	1.81e−3	1.82e−3	–	1.08e−2
	Δ	1.80e−3	1.82e−3	1.80e−3	–	2.78e−1
MOH	IGD	1.82e−3	1.80e−3	1.82e−3	2.78e−1	–
	HV_r	1.81e−3	1.82e−3	2.07e−3	1.08e−2	–
	Δ	1.82e−3	1.81e−3	1.80e−3	2.78e−1	–

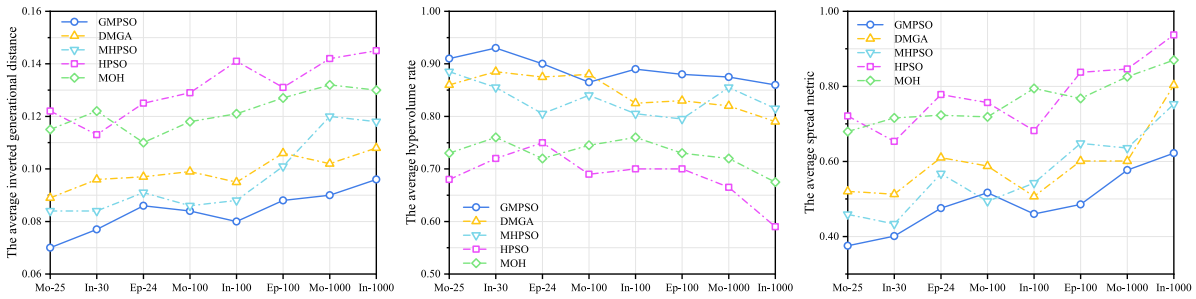
5.4.3. Analysis of results

All algorithms were subjected to 10 independent experiments on each simulation workflow. The results of the experiments are presented in Fig. 11.

On small-scale workflows, all five algorithms achieve good optimization results, but GMPSO obtains a better approximation to the PF. Although the performance on Inspiral-30 is not as good as the individual solutions of DMGA, GMPSO shows the best convergence and uniformity of distribution over the PF. And on Montage-25 and Epigenomics-24, the solutions obtained by GMPSO dominate those obtained by almost all other algorithms.

The dominance of GMPSO is even more pronounced in medium-scale workflows, particularly on Montage-100, where it achieves the best convergence of the solution set. Although the convergence effect on Epigenomics-100 does not show a significant gap compared to DMGA, GMPSO achieves the optimal balance between makespan and energy consumption. On Inspiral-100, GMPSO not only produces the best PF approximation but also demonstrates excellent solution diversity, indicating that it has fully explored the solution space.

In large-scale workflows, the advantages of GMPSO are further revealed, especially on Montage-1000, where the best approximation to the PF is achieved. Moreover, it is obvious that the convergence of the other algorithms decreases as the scale of the workflow increases, which is most intuitive on Inspiral-1000, where only GMPSO achieves better convergence among the five algorithms, and significantly outperforms the other algorithms in terms of energy consumption optimization.



(a) IGD of five algorithms. (b) HV_r of five algorithms. (c) Δ of five algorithms.

Fig. 12. MOP metrics comparison of five algorithms.

To evaluate the performance of our proposed GMPSO algorithm, we use three metrics: IGD, HV, and Δ. For these three metrics, we take the average of 10 independent experiments by taking the concatenated set of different PFs generated by all the tested algorithms and removing the dominated solution as the reference PF. With respect to HV, the worst performing point of the 10 independent experiments is taken as the reference point, and the hypervolume ratio (HV_r) of each algorithm and the reference PF is compared.

As shown in Fig. 12(a), the IGD curve fluctuates during the testing process, but generally shows an increasing trend, which is attributed to the gradual increase in the scale of the workflow and the complexity of the problem. We can observe that GMPSO achieves the best performance among the five algorithms and achieves lower IGD values on workflows of different scales, which indicates that GMPSO has the best convergence. Based on a quantitative analysis of the obtained data, the IGD value achieved by our method is approximately 0.084, outperforming DMGA by 15.28%, MHPSO by 13.08%, HPSO by 35.97%, and MOH by 31.18%.

Fig. 12(b) presents the HV_r curves for all tested algorithms across different workflows, revealing an overall decreasing trend as the problem scale increases. This decline occurs because the quality of algorithmic solutions deteriorates with the growing complexity of the workflow. However, the curve of GMPSO has the least fluctuation, which indicates that our study can perform equally well on large-scale problems and can achieve the solution with the best overall performance. By quantitatively analyzing the data obtained, the HV_r of our method is almost 0.89, which is 5.10% higher than DMGA, 6.84% higher than MHPSO, 29.39% higher than HPSO, and 21.75% higher than MOH.

Fig. 12(c) illustrates the variation curve of the Δ metric. As the workflow scale increases, although the curve exhibits some fluctuations, it demonstrates an overall upward trend. The main reason is that the expansion of the search space increases the difficulty of maintaining diversity and uniformity among solutions, resulting in a higher Δ value. It is evident that our algorithm consistently achieves the best performance across workflows of varying scales, highlighting its strong capability in preserving solution diversity and distribution balance. Based on quantitative analysis, our method attains an average Δ value of approximately 0.49, surpassing DMGA by 17.52%, MHPSO by 13.68%, HPSO by 37.01%, and MOH by 35.80%.

5.4.4. Statistical tests

To evaluate the statistical significance of the performance differences between GMPSO and the benchmark algorithms, we conducted two classical statistical tests.

Specifically, the Wilcoxon signed-rank test was conducted to perform pairwise comparisons between GMPSO and the benchmark algorithms, aiming to determine whether the observed performance improvements are statistically significant. Table 5 presents the test results. It can be observed that across all comparisons between GMPSO and the benchmark algorithms on the three metrics, the p-values are consistently below 0.05. This indicates that the improvements achieved by GMPSO demonstrate significant differences compared to other algorithms, thus confirming the effectiveness of our proposed method.

Table 6
Friedman test on three metrics between algorithms.

Metrics	GMPSO		DMGA		MHPSO		HPSO		MOH	
	V	R	V	R	V	R	V	R	V	R
IGD	8.39e−2	1	9.90e−2	3	9.65e−2	2	1.31e−1	5	1.22e−1	4
HV_r	8.89e−1	1	8.42e−1	2	8.32e−1	3	6.87e−1	5	7.30e−1	4
Δ	4.89e−1	1	5.93e−1	3	5.67e−1	2	7.77e−1	5	7.62e−1	4
Rank	1.00		2.67		2.33		5.00		4.00	

Furthermore, the Friedman test is conducted to further assess the overall ranking and consistency of each algorithm across multiple workflow scenarios. Based on the average metric values (V) of each algorithm across different workflows, their rankings (R) on each metric were determined, which were then used to compute the statistical values. As shown in Table 6, GMPSO ranks first on all metrics, and the computed p-value is 0.0218 (<0.05), indicating that the performance differences among the algorithms are statistically significant, which confirms the overall superiority of GMPSO.

6. Conclusion and future work

Since the traditional scheduling methods are incapable of achieving a comprehensive optimization with better performance due to the heterogeneous type and complexity of the workflow scheduling problem in cloud environments, this study proposes GMPSO. First, the scheduling problem is abstracted into particles and divided into superior, medium and inferior populations according to the fitness, and different genetic operators are integrated on different sub-populations. The superior population accelerates the convergence speed; the inferior population comprehensively explores the solution space to avoid getting trapped in local optima; the medium population ensures a balance between the superior and inferior populations. As the iterations progress, the influence of the genetic operator gradually weakened, eventually transitioning into pure particle swarm operation, which prompts the population to converge more stably on the PF. From the final simulation results, our method obtains the best IGD, HV_r, and Δ values, indicating that GMPSO balances the convergence and diversity of the solutions to achieve the best results in makespan and energy optimization.

GMPSO has good application potential in actual cloud computing scenarios, and is particularly suitable for complex workflow systems that are sensitive to resource utilization, task response time, and energy consumption, such as smart healthcare, scientific computing, and the industrial Internet. While improving scheduling efficiency, this method ensures the diversity of solutions and global optimality, providing an effective tool for multi-objective optimization in cloud environments. However, the current work only considers two QoS indicators, namely latency and energy consumption, and still relies on the traditional evolutionary framework. It has certain limitations when dealing with

highly dynamic and large-scale tasks. As part of future research, we will consider further introducing more QoS objectives (such as cost, reliability, etc.) to more comprehensively reflect the diverse needs in the actual cloud computing environment. Furthermore, we will explore integrating our method with machine learning mechanisms to improve the adaptability and optimization efficiency of the model.

CRediT authorship contribution statement

Peiying Zhang: Writing – original draft, Visualization, Software, Formal analysis, Conceptualization. **Jingfei Gao:** Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Lizhuang Tan:** Supervision, Resources, Project administration, Funding acquisition. **Kai Liu:** Writing – review & editing, Software, Resources, Methodology, Investigation. **Konstantin Igorevich Kostromitin:** Writing – review & editing, Validation, Conceptualization. **Neeraj Kumar:** Writing – review & editing, Supervision, Resources, Formal analysis, Data curation.

Declaration of competing interest

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

Acknowledgments

This work is partially supported by the Natural Science Foundation of Shandong Province, China under Grant ZR2023LZH017, ZR2024MF066, and 2023QF025 (for obtaining experimental data), partially supported by the National Natural Science Foundation of China under Grant 62471493 and 62402257 (for problem model research), partially supported by the Open Foundation of Key Laboratory of Computing Power Network and Information Security, Ministry of Education, Qilu University of Technology (Shandong Academy of Sciences), China under Grant 2023ZD010 (for metrics and model verification).

Availability of data and materials

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

References

[1] M. Kumar, S.C. Sharma, A. Goel, S.P. Singh, A comprehensive survey for scheduling techniques in cloud computing, J. Netw. Comput. Appl. 143 (2019) 1–33.
[2] X. Ma, H. Xu, H. Gao, M. Bian, Real-time multiple-workflow scheduling in cloud environments, IEEE Trans. Netw. Serv. Manag. 18 (4) (2021) 4002–4018.
[3] L. Versluis, A. Iosup, A survey of domains in workflow scheduling in computing infrastructures: Community and keyword analysis, emerging trends, and taxonomies, Future Gener. Comput. Syst. 123 (2021) 156–177.
[4] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, Z. Zeng, Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds, IEEE Trans. Parallel Distrib. Syst. 33 (9) (2021) 2079–2092.
[5] A. Choudhary, I. Gupta, V. Singh, P.K. Jana, A GSA based hybrid algorithm for bi-objective workflow scheduling in cloud computing, Future Gener. Comput. Syst. 83 (2018) 14–26.
[6] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, H. Wang, PIAS: Practical information-agnostic flow scheduling for commodity data centers, IEEE/ACM Trans. Netw. 25 (4) (2017) 1954–1967.
[7] S.-Y. Wang, H.-Y. Fu, Design, implementation, and performance evaluation of an earliest-deadline-first packet scheduling scheme in P4 hardware switches, J. Netw. Comput. Appl. 208 (2022) 103519.
[8] G. Gupta, N. Mangla, An enhanced MIN-MIN algorithm for workflow scheduling in Cloud Computing, in: 2022 4th International Conference on Advances in Computing, Communication Control and Networking, ICAC3N, IEEE, 2022, pp. 2084–2091.
[9] M. Raeisi-Varzaneh, O. Dakkak, Y. Fazea, M.G. Kaosar, Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems, Clust. Comput. 27 (9) (2024) 13407–13419.
[10] S. Tao, Y. Xia, L. Ye, C. Yan, R. Gao, DB-ACO: A deadline-budget constrained ant colony optimization for workflow scheduling in clouds, IEEE Trans. Autom. Sci. Eng. 21 (2) (2023) 1564–1579.
[11] Y. Xie, Y. Sheng, M. Qiu, F. Gui, An adaptive decoding biased random key genetic algorithm for cloud workflow scheduling, Eng. Appl. Artif. Intell. 112 (2022) 104879.
[12] H. Hao, H. Zhu, A self-learning particle swarm optimization for bi-level assembly scheduling of material-sensitive orders, Comput. Ind. Eng. 195 (2024) 110427.
[13] H. Hafsi, H. Gharsellaoui, S. Bouamama, Genetically-modified Multi-objective Particle Swarm Optimization approach for high-performance computing workflow scheduling, Appl. Soft Comput. 122 (2022) 108791.
[14] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, H. Huang, ET2FA: A hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud, IEEE Trans. Serv. Comput. 16 (3) (2022) 1807–1821.
[15] G.J. Mirobi, L. Arockiam, Dynamic workflow scheduling approach for minimizing the response time using an efficient workflow scheduler in cloud computing, in: 2019 International Conference on Smart Systems and Inventive Technology, ICSSIT, IEEE, 2019, pp. 471–476.
[16] E. Cadorel, H. Coullon, J.-M. Menaud, Online multi-user workflow scheduling algorithm for fairness and energy optimization, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID, IEEE, 2020, pp. 569–578.
[17] C. Yin, X. Shi, Fault-tolerant scheduling optimization of cloud workflow based on multi-objective optimization algorithm, in: 2023 China Automation Congress, CAC, IEEE, 2023, pp. 5916–5921.
[18] A. Jayanetti, S. Halgamuge, R. Buyya, Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments, Future Gener. Comput. Syst. 137 (2022) 14–30.
[19] L. Zhang, K. Li, C. Li, K. Li, Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems, Inform. Sci. 379 (2017) 241–256.
[20] H.A. Hassan, S.A. Salem, E.M. Saad, A smart energy and reliability aware scheduling algorithm for workflow execution in DVFS-enabled cloud environment, Future Gener. Comput. Syst. 112 (2020) 431–448.
[21] H. Li, B. Wang, Y. Yuan, M. Zhou, Y. Fan, Y. Xia, Scoring and dynamic hierarchy-based NSGA-II for multiobjective workflow scheduling in the cloud, IEEE Trans. Autom. Sci. Eng. 19 (2) (2021) 982–993.
[22] X. Xia, H. Qiu, X. Xu, Y. Zhang, Multi-objective workflow scheduling based on genetic algorithm in cloud environment, Inform. Sci. 606 (2022) 38–59.
[23] M. Hussain, L.-F. Wei, A. Rehman, M. Ali, S.M. Waqas, F. Abbas, Cost-aware quantum-inspired genetic algorithm for workflow scheduling in hybrid clouds, J. Parallel Distrib. Comput. 191 (2024) 104920.
[24] J. Zhang, L. Cheng, C. Liu, Z. Zhao, Y. Mao, Cost-aware scheduling systems for real-time workflows in cloud: An approach based on Genetic Algorithm and Deep Reinforcement Learning, Expert Syst. Appl. 234 (2023) 120972.
[25] R. Xie, D. Gu, Q. Tang, T. Huang, F.R. Yu, Workflow scheduling using hybrid PSO-GA algorithm in serverless edge computing for the Internet of Things, in: 2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring), IEEE, 2022, pp. 1–7.
[26] X. Tang, C. Shi, T. Deng, Z. Wu, L. Yang, Parallel random matrix particle swarm optimization scheduling algorithms with budget constraints on cloud computing systems, Appl. Soft Comput. 113 (2021) 107914.
[27] Y. Xie, Y. Zhu, Y. Wang, Y. Cheng, R. Xu, A.S. Sani, D. Yuan, Y. Yang, A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment, Future Gener. Comput. Syst. 97 (2019) 361–378.
[28] L. Yang, Y. Xia, L. Ye, R. Gao, Y. Zhan, A fully hybrid algorithm for deadline constrained workflow scheduling in clouds, IEEE Trans. Cloud Comput. 11 (3) (2023) 3197–3210.
[29] S. Bansal, H. Aggarwal, A multiobjective optimization of task workflow scheduling using hybridization of PSO and WOA algorithms in cloud-fog computing, Clust. Comput. 27 (8) (2024) 10921–10952.
[30] C. Lu, J. Zhu, H. Huang, Y. Sun, A multi-hierarchy particle swarm optimization-based algorithm for cloud workflow scheduling, Future Gener. Comput. Syst. 153 (2024) 125–138.
[31] H. Li, D. Wang, M. Zhou, Y. Fan, Y. Xia, Multi-swarm co-evolution based hybrid intelligent optimization for bi-objective multi-workflow scheduling in the cloud, IEEE Trans. Parallel Distrib. Syst. 33 (9) (2021) 2183–2197.
[32] J. Xu, J. Yang, P. Li, Z. Wang, C. Huang, X. Yao, A knowledge guided multi-population evolutionary algorithm for dynamic workflow scheduling problem, in: 2024 IEEE Conference on Artificial Intelligence, CAI, IEEE, 2024, pp. 21–28.
[33] H. Qiu, X. Xia, Y. Li, X. Deng, A dynamic multipopulation genetic algorithm for multiobjective workflow scheduling based on the longest common sequence, Swarm Evol. Comput. 78 (2023) 101291.
[34] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[35] W. Chen, E. Deelman, Workflowsim: A toolkit for simulating scientific workflows in distributed environments, in: 2012 IEEE 8th International Conference on E-Science, IEEE, 2012, pp. 1–8.

[36] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw.: Pr. Exp. 41 (1) (2011) 23–50.

[37] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2013) 682–692.

[38] A. Verma, S. Kaushal, A hybrid multi-objective particle swarm optimization for scientific workflow scheduling, Parallel Comput. 62 (2017) 1–19.

[39] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, M. Chen, Cost and makespan-aware workflow scheduling in hybrid clouds, J. Syst. Archit. 100 (2019) 101631.