Future Generation Computer Systems 175 (2026) 108118

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Adaptive-oriented mutation snake optimizer for scheduling budget-constrained workflows in heterogeneous cloud environments

Yanfen Zhang[a], Longxin Zhang[a,1,*], Buqing Cao[a,b], Jing Liu[c], Wenyu Zhao[a], Jianguo Chen[d], Keqin Li[e]

[a] School of Computer Science, Hunan University of Technology, 412007, Zhuzhou, China
[b] Key Laboratory of Intelligent Sensing System and Security (Hubei University), Ministry of Education, 430062, Wuhan, China
[c] Department of Computer Science and Technology, Wuhan University of Science and Technology, 430081, Wuhan, China
[d] School of Software Engineering, Sun Yat-Sen University, 519082, Zhuhai, China
[e] Department of Computer Science, State University of New York, 12561, New Paltz, NY, USA

ARTICLE INFO

Keywords:
Budget
Heterogeneous cloud computing
Makespan
Snake optimizer
Workflow scheduling

ABSTRACT

Cloud computing, recognized as an advanced computing paradigm, facilitates flexible and efficient resource management and service delivery through virtualization and resource sharing. However, the computational capabilities of resources in heterogeneous cloud environments are often correlated with their costs; thus, budget constraints are imposed on users who require rapid response times. We introduce a novel metaheuristic optimization algorithm called the snake optimizer (SO), which is aimed at workflow scheduling in cloud environments, to tackle the challenge mentioned. We also integrate random mutation to enhance the algorithm's global search capability to overcome the limitation of SO's being prone to local optima. Additionally, we aim to increase the success rate of finding feasible solutions within budget constraints; thus, we implement a directional strategy to guide the evolutionary paths of the snake individuals. In this context, excessive randomness and overly rigid directionality can adversely affect the algorithm's search performance. We propose an adaptive-oriented mutation (AOM) mechanism to balance the two aspects mentioned. This AOM mechanism is integrated with SO to create AOM-SO, which effectively addresses the makespan minimization problem for workflow scheduling under budget constraints in heterogeneous cloud environments. Comparative experiments using real-world scientific workflows show that AOM-SO achieves a 100 % success rate in identifying feasible solutions. Moreover, compared with the state-of-the-art algorithms, it reduces makespan by an average of 43.03 %.

1. Introduction

The rapid advancement of Internet technologies has resulted in the emergence of cloud computing as a novel paradigm for delivering dynamic, reliable, and elastic computing services [1]. This innovative technology transforms how global users share computing resources over the Internet and serves as a significant catalyst for the advancement of cutting-edge technologies, such as 5G, the Internet of Things, and the industrial Internet. Thus, cloud computing has become an indispensable support system [2]. As a fundamental component of cloud computing, virtualization technology enables dynamic resource scheduling and scaling. Numerous computing nodes with varying capabilities and characteristics are effectively integrated through virtualization to create a vast heterogeneous cloud resource pool. This setup enables users to access and utilize computing resources flexibly from the pool through a pay-per-use payment model over the Internet [3]. Cloud computing is particularly well-suited for addressing compute-intensive scientific workflows [4]. It has been extensively applied to address the challenges posed by large-scale data and complex workflows across various domains, including biology, physics, and astronomy [5].

The workflow scheduling problem is particularly critical in cloud computing environments. Workflow scheduling algorithms in cloud centers should focus on determining the optimal mapping between a set of interdependent tasks and heterogeneous virtual resources to achieve the users' specified objectives, whether they involve minimization or maximization. However, workflow scheduling is classified as an NP-hard problem [6]. Thus, achieving optimal scheduling becomes challenging. Researchers typically investigate the workflow scheduling problem

* Corresponding author.
E-mail addresses: yanfen.z@stu.hut.edu.cn (Y. Zhang), longxinzhang@hut.edu.cn (L. Zhang), buqingcao@hut.edu.cn (B. Cao), luijing_cs@wust.edu.cn (J. Liu), zhaowenyu@hut.edu.cn (W. Zhao), chenjg33@mail.sysu.edu.cn (J. Chen), lik@newpaltz.edu (K. Li).
1 Tel.: +86-731-22183345.

https://doi.org/10.1016/j.future.2025.108118
Received 26 January 2025; Received in revised form 25 August 2025; Accepted 1 September 2025
Available online 5 September 2025

under user-defined quality of service (QoS) constraints [7]. Existing studies primarily focus on optimizing objectives, such as completion time (makespan), cost, and reliability, to satisfy QoS constraints related to deadlines, energy consumption, and budget. As a result, efficient and approximately optimal scheduling solutions can be thereby proposed.

In recent years, due to the data-intensive or compute-intensive characteristics of most workflow applications [8], the makespan of workflows has become an important metric of great concern, and users expect workflows to complete execution as soon as possible. However, the computing resources in cloud computing centers are usually provided to users in the form of virtual machines (VMs), and the computing speed of VMs is positively correlated with their prices, and high-performance VMs often imply higher expenses. Excessive expenses will undoubtedly have a negative impact on user satisfaction, making users face the dilemma of cost-performance trade-off when making resource choices. Compared to pursuing the minimization of execution cost, this study tends to set budget limits for the entire workflow. Related studies [9,10] have also explicitly proposed limiting the cost of renting computing resources to ensure efficient execution within a cost-controlled framework. The budget-constrained workflow scheduling problem is formulated in the context of a user submitting a workflow to a cloud scheduling platform with a predefined budget. In this context, the core challenge of the cloud scheduling platform is to provide a feasible and efficient scheduling solution. The solution must not only adhere to budget constraints but also minimize the makespan of the workflow to optimize both resource utilization and execution efficiency. Effectively solving this problem can effectively improve the operational efficiency of the cloud scheduling platform and meet the needs of users, thus enhancing user satisfaction.

Currently, the dominant solutions for minimizing workflow makespan under budget constraints include heuristic, metaheuristic, and deep reinforcement learning (DRL)-based [11] scheduling algorithms. Heuristic algorithms can quickly generate workflow scheduling strategies based on expert knowledge; however, their effectiveness is primarily limited to small-scale workflows, and they often depend on relatively simple rules, thereby making them vulnerable to local optima. Furthermore, their ability to find optimal solutions in dynamic cloud environments is limited [12]. DRL-based approaches are still in their early stages of development, and various challenges are encountered when scheduling problems for large-scale complex applications, such as the intricate design of reward functions, difficulties in hyperparameter tuning, and inadequate exploration efficiency, are tackled [13]. By contrast, metaheuristic algorithms have been extensively employed in large-scale workflows to find approximately optimal scheduling solutions.

Existing metaheuristic scheduling algorithms typically employ random mutations to enhance population diversity, thereby reducing the risk of becoming trapped in local optima. Li et al. [14] conducted optimization research by employing the owl search algorithm (OSA). They also modified the population update mechanism of this algorithm to explore an enhanced variant known as the OSA with a newly designed mutation strategy (OSAM), specifically tailored for workflow scheduling to minimize makespan under budget constraints. OSAM incorporates a mutation operator into OSA to enhance its global search capabilities and reduce the chances of being trapped in local optima. Additionally, Li et al. [8] proposed the mutation-driven and population grouping poor and rich optimization algorithm (MG-PRO) for scheduling workflows with budget constraints in the cloud to minimize the makespan. MG-PRO features an evolution-aware mutation strategy that intervenes in the evolutionary process to enhance population diversity. However, random mutations can lead to a directionless random walk behavior in the position update process of individuals within the population, thereby complicating the search path and increasing the computational challenges associated with reaching the global optimum. In particular, the aimless exploration caused by random mutations may reduce the algorithm's efficiency in effectively identifying feasible solutions within a constrained search budget when resources (such as computational time and the number of iterations) are limited. Moreover, the reliance on random mutations can impair the algorithm's performance in minimizing makespan because these aimless attempts often fail to target effectively the areas of the solution space that can significantly enhance makespan optimization. In summary, random mutations play a vital role in improving the exploratory capabilities of metaheuristic algorithms; however, the challenges they introduce cannot be overlooked. The balance between randomness and directionality must be explored to achieve dual optimization of search efficiency and solution quality, particularly when seeking efficient solutions to complex optimization problems.

The snake optimizer (SO) [15] is an innovative metaheuristic optimization algorithm that greatly surpasses existing approaches in workflow scheduling. Given that it has demonstrated performance advantages, this study asserts that SO has significant potential for tackling complex workflow scheduling challenges. Therefore, we apply SO to workflow scheduling problems in heterogeneous cloud computing while investigating a population update mechanism that combines randomness with orientation. The study concentrates on two main areas: (1) The incorporation of random mutations aims to overcome the original SO algorithm's limitation of often converging on local optima. However, excessive randomness can lead to the inefficient use of computational resources, increase computational complexity, and hinder the algorithm's ability to reach the global optimum effectively. (2) The development of an oriented strategy aims to mitigate the drawbacks of random mutations while enhancing the algorithm's success rate in finding feasible solutions within budget constraints. Nevertheless, excessive orientation may also result in the algorithm becoming trapped in local optima.

This study investigates an adaptive-oriented mutation (AOM) mechanism that balances randomness and orientation. Then, it is integrated with SO to develop a scheduling algorithm referred to as the AOM-SO. The AOM-SO addresses workflow scheduling challenges under budget constraints in heterogeneous cloud computing environments by thoroughly evaluating the heterogeneity among different VMs, the dependencies of workflow tasks, and budget limitations. This approach aims to identify optimal or near-optimal scheduling solutions that minimize the makespan of the workflow. The contributions of this study are as follows:

- The integration of SO into the workflow scheduling problem under budget constraints simulates the evolutionary stages of snakes to approximate a globally optimal scheduling solution.
- The proposed AOM mechanism balances randomness and orientation to address SO's tendency to converge on local optima. This mechanism enhances the evolutionary trajectory of snake individuals through AOMs across multiple dimensions, thereby improving the success rate in identifying feasible solutions that meet budget constraints.
- The AOM-SO approach is introduced with a series of comparative experimental results, demonstrating that this approach achieves a high success rate in finding feasible solutions and significantly outperforms similar algorithms in reducing makespan.

A shorter version of this work was accepted at the IEEE ISPA conference in 2024 [16]. However, that version focused solely on the positional offset strategy of snake individuals in a single dimension, neglecting the challenge of the algorithm's high-dimensional complexity in large-scale workflows with numerous tasks. This extended version addresses this issue by integrating the total number of tasks in a workflow into the algorithm's execution logic and introducing the AOM mechanism to enhance its capability in managing high-dimensional problems. Additionally, this study offers a more comprehensive discussion in the introduction and related work review. It enhances the algorithm section with intuitive diagrams of the algorithmic structure and includes a deeper analysis of the algorithm's time complexity. It also references additional comparative algorithms and performance metrics and incorporates two types of real-world scientific workflow datasets beyond those

Table 1
A comparison of various related scheduling methods.

Ref	Constraints	Objectives	Characteristics	Challenges and limitations
[17]	Budget	Makespan	Remaining cheapest budget	It is sensitive to task prioritization, which may adversely affect the scheduling of low-priority tasks.
[18]	Budget	Makespan	Normalization	It needs further enhancement for constraint satisfaction.
[10]	Budget	Makespan	Budget reallocation	It does not consider the heterogeneity of VMs.
[19]	Budget, Deadline	Makespan, Cost	Improved Archimedes optimization algorithm	It is strongly influenced by the parameter settings of the initial population.
[20]	Budget, Deadline	Cost	Improved ant colony optimization algorithm	It is not suitable for scheduling under relaxed constraints.
[21]	–	Makespan	Heterogeneous earliest-finish-time	It aims to balance the average makespan with the number of available processors.
[22]	–	Makespan	Improved predict priority	Its effectiveness is limited on Epigenomic and comparable graphs.
[23]	Budget	Makespan	Adaptive iterated local search	It has restricted QoS metrics.
[24]	Budget	Makespan	Greedy resource provisioning	It does not ensure adherence to strict constraints.
[25]	–	Makespan	Improved optimistic cost matrix	It does not consider budget constraints.
[26]	–	Makespan	Prediction of makespan matrix	It does not consider budget constraints.
[27]	–	Makespan	Improved genetic algorithm	It does not consider budget constraints.
[28]	–	Makespan, Cost	Improved poor and rich optimization algorithm	It has a high dependence on population parameters.
[29]	–	Makespan, Cost	Improved ant colony optimization algorithm	It needs to improve runtime performance.
Our work	Budget	Makespan	Adaptive-oriented mutation snake optimizer	It does not take dynamic workflows into consideration.

used in the conference version, enabling a more comprehensive experimental evaluation.

This study is organized as follows. Section 2 reviews recent advancements in research concerning workflow scheduling problems. Section 3 offers a comprehensive description of the workflow scheduling model by highlighting the constraints and optimization objectives of the study. Section 4 introduces the workflow scheduling algorithm proposed herein. Section 5 outlines the experimental setup and analyzes the results. Finally, Section 6 summarizes the contributions of this study and discusses potential directions for future research.

2. Related work

Workflow scheduling has been thoroughly explored within the field of cloud computing. This section offers a comprehensive review of the pertinent literature, categorizing the research into three main types: (1) budget-constrained workflow scheduling, (2) time-optimized workflow scheduling, and (3) multiobjective workflow scheduling. Table 1 provides a summary and comparison of the relevant literature.

2.1. *Budget-constrained workflow scheduling*

In recent years, numerous researchers have extensively studied budget-constrained workflow scheduling issues. Arabnejad and Barbosa [17] developed a heterogeneous budget-constrained scheduling algorithm that utilizes the remaining budget to ensure that workflow execution costs stay within the allocated budget. This approach effectively addresses the challenges associated with budget-constrained workflow scheduling. However, the rules governing the variation of the remaining budget affect task execution based on priority levels, thereby resulting in fairness issues for low-priority tasks. These issues negatively impact the overall workflow scheduling performance. Chen et al. [30] proposed an efficient task budgeting algorithm based on budget-level allocation to tackle the challenges mentioned. Their main idea is to shift budget constraints from the workflow level to individual tasks, thereby reducing the impact of task priority on task budgets. Kalyan et al. [18] investigated a workflow scheduling algorithm that operates under normalized budget constraints. They mapped workflow tasks to VMs by calculating the expected fair budget to achieve the earliest completion times while adhering to fair budget limits. This approach avoids the selection of VMs with excessively high costs that can negatively affect the budgets of unscheduled tasks. While existing heuristics address the budget-constrained workflow scheduling problem effectively, they typically lack a deterministic fine-tuning mechanism and subsequent optimization techniques after generating the initial scheduling plan. Recent research has introduced improvements in this area. Fan et al. [9] introduced a priority adjuster and a critical task optimizer, enabling dynamic priority adjustment and resource allocation optimization through in-depth task characteristic analysis. Zhang et al. [10] reduced the critical path length by iteratively adjusting task-resource mappings under limited budgets, leading to significant decreases in makespan, although at the cost of lower computational efficiency in a heterogeneous VM environment. Wu et al. [31] employed a dynamic remaining budget reallocation strategy to enable flexible resource combinations by prioritizing budget allocation to tasks with the highest utility. In the field of metaheuristic algorithms, Kushwaha and Singh [19] enhanced a metaheuristic scheduling algorithm based on the Archimedean optimization method by incorporating both budget and deadline constraints. While this approach has lower computational complexity, its performance sensitivity to initial population settings and parameter configurations is notable. Tao et al. [20] proposed a hybrid optimization framework that combines ant colony optimization and heuristic rules to achieve flexible resource combinations in heterogeneous VM environments. This method performs well under strict constraints but demonstrates limited optimization effects in scenarios with looser constraints.

Although the aforementioned scheduling algorithms have achieved some success in addressing workflow scheduling issues under budget constraints, they struggle to adapt to varying budget limitations because of the absence of a dynamic budget-aware mechanism to inform resource allocation. As a result, their performance may diminish as the scale of workflow tasks increases.

2.2. *Time-optimized workflow scheduling*

Topcuoglu et al. [21] introduced the well-known heterogeneous earliest finish time (HEFT) algorithm for heterogeneous processors, which has significantly contributed to research in makespan optimization methods for workflow scheduling. Inspired by this work, numerous researchers have explored various effective strategies aimed at minimizing makespan. Djigal et al. [22] devised an innovative lookahead mechanism during the task priority sorting phase. This mechanism is designed to schedule tasks on heterogeneous processors efficiently and reduce task execution time. Qin et al. [23] designed a novel adaptive iterative local search framework that combines a greedy resource allocation scheme with the HEFT algorithm to generate an initial solution. This framework incorporates a novel reinforcement strategy and an adaptive penalty function aimed at directing the search process toward the edges of the feasible solution space. As a result, the makespan can be optimized. Faragardi et al. [24] made targeted modifications and enhancements to the traditional HEFT algorithm by proposing a new greedy scheduling algorithm focused on minimizing the makespan of a given workflow. Wang et al. [25] proposed a list scheduling algorithm known as the average earliest finish time (AEFT), which utilizes an enhanced optimistic cost matrix. The AEFT algorithm takes into account the out-degree property of the task and integrates it with the workflow topology to minimize overall scheduling time. By employing an insertion policy to leverage idle time slots in the processor, the algorithm aims to optimize resource utilization. Zhang et al. [26] proposed a priority calculation algorithm based on a predicted makespan matrix that maximizes the reduction of execution time for workflow applications while satisfying priority constraints. The proposed algorithm is suitable for efficient workflow scheduling in heterogeneous cloud environments. Jiang et al. [27] introduced a novel coding method using real numbers to signify the relative positions of tasks within the set of schedulable tasks. This innovation led to the development of a real-relative coded genetic algorithm (GA), which effectively resolves issues encountered in traditional and existing real-coded GAs. Specifically, this new algorithm addresses challenges related to violations of task priority constraints and the constrained range of gene values.

However, the greedy decisions made by heuristic methods in the early stages of scheduling can significantly impact subsequent resource allocation, potentially resulting in locally optimal scheduling strategies, particularly with large-scale workflows. Although metaheuristic methods are commonly used for large-scale workflow scheduling, they too can become stuck in local optimum solutions. Consequently, further exploration and breakthroughs are urgently needed to address the global complexities of workflow scheduling problems and minimize scheduling lengths.

2.3. Multiobjective workflow scheduling

Given the complexity of workflow scheduling problems and their diverse requirements, some researchers have started to investigate multiobjective optimization algorithms [32] to find additional comprehensive and effective solutions for the increasingly intricate and dynamic challenges of workflow scheduling. The robust search capabilities and flexibility of metaheuristics offer a promising approach for tackling multiobjective workflow scheduling issues. Li et al. [28] introduced a bi-objective optimization algorithm that incorporates coevolution and elite learning to minimize makespan and workflow costs. Rathi et al. [33] formalized the workflow scheduling problem as a multiobjective optimization challenge within a GA framework by considering execution time and communication costs. They also implemented a fitness-dependent optimizer inspired by the reproductive behavior of bees to enhance the optimization process. Chen et al. [29] modeled the workflow scheduling challenge as a multiobjective optimization problem by focusing on the simultaneous optimization of execution time and cost. They proposed a novel multiobjective ant colony system based on a multipopulation coevolution framework, where time ants and cost ants independently manage the optimization processes for their respective objectives. Guo et al. [34] proposed a dynamic neighborhood grouping strategy based on data dependencies between workflow tasks. They designed new crossover and mutation operators to explore feasible solutions and thus improve search efficiency, aiming to simultaneously optimize the makespan, cost, and energy consumption of workflow execution. Reddy and Reddy [35] developed a multi-objective task scheduling algorithm within a reinforcement learning framework. This algorithm efficiently selects VMs using a fuzzy self-defense algorithm to map workflows to suitable VMs. The primary objectives are to optimize makespan, cost, and energy consumption while also balancing user and provider requirements. Recognizing the varied needs of users, the authors in [36–38] explored multiobjective optimization algorithms under QoS constraints, such as deadlines and budgets. Furthermore, the integration of metaheuristic algorithms with DRL has provided new insights into the field of workflow scheduling. Zhang et al. [12] investigated a real-time workflow scheduling method that combines GA with DRL to reduce execution costs and response times.

In general, optimizing for makespan often requires increasing costs, whereas reducing costs can result in long scheduling times. Thus, multiobjective workflow scheduling must reasonably balance the potentially conflicting optimization objectives, such as time and cost. In contrast to studies concentrating on multiobjective workflow scheduling problems, this study concentrates on single-objective optimization, which aims to minimize the scheduling length of workflows. In this context, cost is treated as a significant constraint. This study aims to minimize scheduling length while adhering to budget constraints.

3. Models and problem formulation

This section presents the models related to cloud resources, workflows, and costs. Then, the optimization problem is described. Additionally, Table 2 offers a summary of the key notations used in this study.

3.1. Cloud resource model

The computational resources of a cloud computing center can be represented as a set of VM instances, denoted as $V = \{I_1, \dots, I_k, \dots, I_{|V|}\}$, where $|V|$ represents the total number of VM instances. In a heterogeneous cloud environment, VM instances are available in various types and configurations of resources (such as CPU, memory, and network) [5], thereby allowing users to rent any quantity at any time via the Internet according to their needs. All VMs are assumed to be provisioned by the same data center and share a common average bandwidth [13].

A VM instance possesses the following characteristics [5,39]:

$$I_k = \left\{ ID_k, Type_k, p_k, \eta_k \right\}, \tag{1}$$

where ID_k denotes the k-th VM instance, $Type_k$ indicates the type of VM I_k, p_k represents the unit price of VM I_k, and η_k indicates the processing speed of VM I_k.

3.2. Workflow application model

In cloud computing systems, a workflow application is represented as a directed acyclic graph (DAG) [40], where tasks with data dependencies are depicted as nodes, and the dependencies between tasks are illustrated by edges. In this study, $G = \{T, E, U, W\}$ is used to signify a workflow application.

(1) $T = \{t_1, \dots, t_i, \dots, t_{|T|}\}$ denotes a set of tasks within a workflow, and $|T|$ represents the total number of tasks in the workflow.

(2) $E = \{e_{i,j} | t_i, t_j \in T\}$ refers to the collection of communication edges in DAG, which illustrate data dependencies. Each edge signifies a constraint on task execution, indicating that task t_i must be completed before task t_j can begin.

(3) $c_{i,j} \in U$ represents the communication time required to transfer relevant data from task t_i to task t_j [41]. It can be described as follows:

$$c_{i,j} = TD_{i,j} / bw, \tag{2}$$

where $TD_{i,j}$ indicates the data transmitted from task t_i to task t_j, and bw signifies the average transmission bandwidth. If task t_i and task t_j are assigned to the same VM for execution, then $c_{i,j} = 0$.

(4) $w_{i,k} \in W$ represents the time required for task t_i to execute on VM I_k [41]. It can be calculated by the following expression:

$$w_{i,k} = TS_i / \eta_k, \tag{3}$$

where TS_i indicates the size for task t_i.

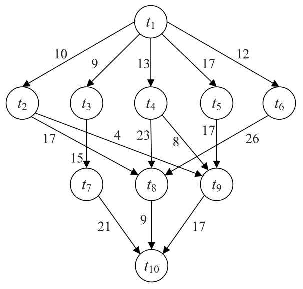
(5) The set of direct predecessor tasks, $pred(t_i)$, and the set of direct successor tasks, $succ(t_i)$, for task t_i are defined as follows:

$$pred(t_i) = \{t_\zeta | e_{\zeta,i} \in E, t_\zeta \in T\}, \tag{4}$$

$$succ(t_i) = \{t_\vartheta | e_{i,\vartheta} \in E, t_\vartheta \in T\}. \tag{5}$$

Table 2
Notations and definitions.

Notation	Definition
V	A set of VMs
$\|V\|$	The number of VMs
I_k	k-th VM
p_k	Price of VM I_k
η_k	Process speed of VM I_k
$T(E)$	A set of workflow tasks (edges)
t_i	i-th task
$\|T\|$	The number of workflow tasks
$c_{i,j}(TD_{i,j})$	Transfer time (data) between task t_i and task t_j
bw	Transfer bandwidth between VMs
$w_{i,k}$	Execution time of task t_i on VM I_k
$s_i(f_i)$	Start (finish) time of task t_i
$M(C)$	Makespan (total execution cost) of a workflow
B	Budget constraint for a given workflow
$\mathbb{S}'$	Current population
$\mathbb{S}''$	Updated population from $\mathbb{S}'$
$\mathbb{S}'''$	Updated population by comparing $\mathbb{S}'$ and $\mathbb{S}''$
$\mathbb{S}^{l+1}$	Updated population from $\mathbb{S}'''$
δ, $\delta_m(\delta_f)$	Population size and the number of males (females)
$X'(X'', X''')$	An individual in the population $\mathbb{S}'(\mathbb{S}'', \mathbb{S}''')$
y	Index of the individual
x_i	Index of the VM in V allocated to task t_i
$UB(LB)$	Maximum (minimum) number of the available VMs for workflow tasks
$X_{UB}(X_{LB})$	Upper (lower) boundary of the solution space
Q	Food quantity
$Temp$	Ambient temperature
$l(L)$	Current (maximum) number of iterations
c_1, c_2, c_3	Step size at different evolutionary stages in SO
X^*, $X^*_m(X^*_f)$	The best individual and the best male (female) individual
E_m, E_f, F_m, F_f, M_m, M_f	Snake's update ability at different evolutionary stages
μ	Oriented probability of the AOM module
$\tilde{B}$	Negative budget cost for an individual violating budget constraint
Υ_i	Adaptiveness of task t_i
$subB(t_i)$	Dynamic sub-budget for task t_i
D	A set of random dimensions
$\|D\|$	The number of randomly selected dimensions
$q(d_r)$	Mutation queue for d_r-th dimension
Ω	A set of mutation queues for multiple dimensions
β	Budget constraint factor



(a) A typical workflow application

Task	I_1	I_2	I_3
t_1	13	10	7
t_2	19	14	10
t_3	17	13	9
t_4	21	16	12
t_5	14	10	8
t_6	20	15	11
t_7	22	17	13
t_8	18	13	10
t_9	23	18	13
t_{10}	15	11	8

(b) Execution time of tasks on different VMs

Fig. 1. A sample workflow graph with 10 tasks.

In a DAG, a task that lacks predecessor tasks is referred to as an entry task (t_{entry}), whereas a task without successor tasks is referred to as an exit task (t_{exit}). Fig. 1(a) displays a typical workflow DAG graph consisting of 10 task nodes, while Fig. 1(b) illustrates the execution times of these 10 tasks on various VMs.

In this study, the start time and finish time of task t_i executing on VM I_k are denoted as s_i and f_i, respectively. The corresponding expressions for these calculations are as follows:

$$s_i = \max\left\{a_k, \max_{t_\zeta \in pred(t_i)}\left\{f_\zeta + c_{\zeta,i}\right\}\right\}, \tag{6}$$

$$f_i = s_i + w_{i,k}, \tag{7}$$

where a_k represents the available time of VM I_k. Therefore, $s_{entry} = a_k$ is for an entry task t_{entry}.

The makespan M of the workflow G refers to the total time required from the start of the workflow scheduling to its completion. Its calculation can be expressed as follows:

$$M(G) = \max_{t_i \in T}\left\{f_i\right\}. \tag{8}$$

3.3. Cost model

The cost model is founded on a pay-per-use mechanism [41]. Given that most cloud providers bill exclusively for the resources consumed and do not charge for internal data transfers, this model posits that the cost linked to a VM is based solely on the execution time while the VM is operational. As a result, this study does not consider costs associated with internal data transfers, meaning users are only required to pay for the time they actually use the VM [42]. Given the heterogeneity of VMs, we define the execution cost associated with running task t_i on VM I_k as follows:

$$C_{exe}(t_i, I_k) = w_{i,k} \times p_k. \tag{9}$$

Therefore, the total cost $C(G)$ of workflow G can be estimated by

$$C(G) = \sum_{i=1}^{|T|} C_{exe}\left(t_i, I_k\right). \tag{10}$$

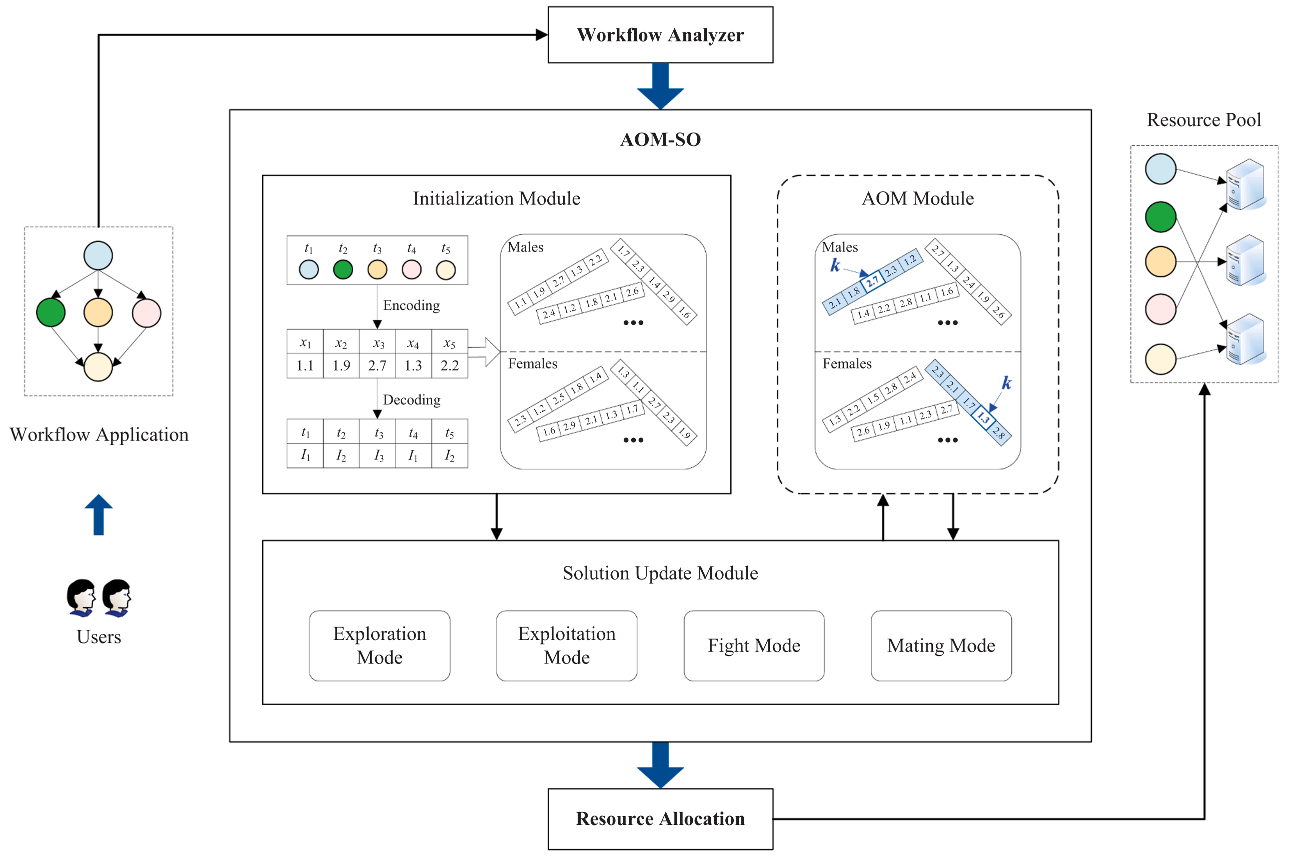


Fig. 2. Workflow scheduling process by AOM-SO in clouds.

3.4. Problem formulation

The scheduling problem explored in this study focuses on scheduling workflows with budget constraints within heterogeneous cloud computing systems to minimize the makespan of the workflows. In particular, let $B(G)$ represent the budget constraint defined by the user. The proposed algorithm seeks to determine the optimal mapping of a set of interdependent tasks to a collection of heterogeneous VMs while ensuring compliance with the budget constraint. In this way, the workflow's makespan is minimized.

The optimization problem is formulated as follows:

$$Minimize: M(G), \tag{11}$$

$$subject\ to: C(G) \leq B(G). \tag{12}$$

4. AOM-SO algorithm

This section delivers a comprehensive overview of the AOM-SO algorithm presented in this study. Fig. 2 depicts the workflow scheduling process employed by the AOM-SO algorithm. The AOM-SO algorithm consists of three distinct modules: the initialization module, the solution update module, and the AOM module. Furthermore, the specific details of the AOM-SO algorithm are provided in Algorithms 1–3. This section defines essential concepts, clarifies the operational mechanisms of each module through mathematical expressions, illustrates the principles of their collaborative operation, and analyzes the algorithm's time complexity to improve the understanding of the AOM-SO algorithm.

4.1. Initialization module

Similar to original SO algorithm [15], the AOM-SO algorithm initiates its optimization process by generating a randomly distributed initial population within the solution space. This population serves as the foundational starting point for the search.

Step 1: Generate the initial population $\mathbb{S}' = \{X'_1, \ldots, X'_y, \ldots, X'_\delta\}$, where δ is the population size.

The individual $X'_y = \{x_1, \ldots, x_i, \ldots, x_{|T|}\}$ in the population signifies a potentially viable solution to the optimization problem (i.e., a workflow scheduling scheme). It captures the mapping relationship between workflow tasks and heterogeneous VMs. As shown in the initialization module of Fig. 2, the encoding and decoding processes indicate that the dimension of the individual, denoted as Dim, is equal to the number of tasks $|T|$ in workflow G. Each dimension value x_i is a floating-point number that corresponds to the index of the VM assigned to task t_i after a rounding function is applied. For instance, $x_3 = 2.7$ signifies that task t_3 is allocated to VM I_3 for execution. The available values for each dimension depend on the number of VMs $|V|$. The calculation expression is as follows:

$$x_i = LB + \tau \times (UB - LB), \tag{13}$$

where τ represents a random number between 0 and 1, and LB and UB indicate the minimum and maximum indexes of the available VMs, respectively.

The original SO algorithm is inspired by the behavior of snakes. It considers the unique traits of male and female snakes and highlights the differences in gender during the simulation of the evolutionary optimization process.

Step 2: Partition the initial population $\mathbb{S}'$ into two distinct groups: males and females.

The number of individuals in the male group is δ_m, and the number of individuals in the female group is δ_f. Their expression can be computed by Eqs. (14) and (15) [15].

$$\delta_m \approx \frac{1}{2}\delta, \tag{14}$$

$$\delta_f = \delta - \delta_m. \tag{15}$$

The behavioral activities of male and female snakes are influenced by temperature and significantly by the availability of food. In particular, mating activities occur only when food resources are plentiful and environmental temperatures are relatively low.

Step 3: Define the parameters Q and $Temp$ to model the fluctuations in food availability and temperature within the environment.

As the algorithm advances through its iterations, the amount of food steadily rises while the temperature gradually falls, as outlined in the following:

$$Q = c_1 \times \exp\left(\frac{l-L}{L}\right), \tag{16}$$

$$Temp = \exp\left(\frac{-l}{L}\right), \tag{17}$$

where l denotes the current iteration number, L represents the maximum iteration number, c_1 is a constant equal to 0.5, and $\exp(\cdot)$ indicates the exponential function. Furthermore, Q and $Temp$ signify the quantity of food and the temperature in the environment at the l-th iteration of the algorithm, respectively.

The boundary parameters for the quantity of food and temperature are set as $\Delta_Q = 0.25$ and $\Delta_T = 0.6$, respectively. These parameters are designed to characterize the state of the algorithm's optimization process and support the population update mechanism in its pursuit of a globally optimal solution.

Algorithm 1 AOM-SO.

```
Input: $\delta$, $L$, $Dim$, $X_{LB}$, $X_{UB}$, $G$, $B(G)$, and $V$.
Output: $X^*$.
 1: Initialize a random population $\mathbb{S}'$ and partitioned it into male and
    female groups;
 2: for $l \leftarrow 1$ to $L$ do
 3:     Locate the best male individual $X_m^*$ and the best female individual
        $X_f^*$;
 4:     Calculate the current iteration values of $Q$ and $Temp$ by using Eqs.
        (16) and (17);
 5:     $\mathbb{S}'' \leftarrow$ Update $\mathbb{S}'$ by using Algorithm 2;
 6:     for each $X_y''$ in $\mathbb{S}''$ do
 7:         for $i \leftarrow 1$ to $|T|$ do
 8:             Adjust $1 \le x_i \le |V|$;
 9:         end for
10:     end for
11:     $\mathbb{S}''' \leftarrow$ Compare $\mathbb{S}'$ and $\mathbb{S}''$ based on the updated rules;
12:     $\mathbb{S}^{l+1} \leftarrow$ Renew $\mathbb{S}'''$ by using Algorithm 3;
13: end for
14: return $X^*$.
```

4.2. Solution update module

The main challenge in solving optimization problems with metaheuristic algorithms lies in effectively updating the population's positions to explore and converge on the optimal solution. In the scheduling problem examined in this study, the solution update module employs the population updating mechanism of the original SO algorithm, guided by two main goals: first, to identify solutions that adhere to budget constraints, and second, to minimize the makespan based on those solutions.

Rules. For any two solutions X_1 and X_2, X_1 is considered superior to X_2 if and only if the following conditions are met:

$$\begin{cases} M(X_1) < M(X_2), & \text{if } C(X_1), C(X_2) \le B(G); \\ C(X_1) < C(X_2), & \text{if } C(X_1), C(X_2) > B(G); \\ C(X_1) < B(G), & \text{otherwise}, \end{cases} \tag{18}$$

where $M(\cdot)$ denotes the makespan of the workflow scheduling scheme, and $C(\cdot)$ represents the total cost of the workflow scheduling scheme.

The solution update comprises four population update modes: exploration mode, exploitation mode, fight mode, and mating mode. In each iteration, one of these modes is chosen to update the parent population $\mathbb{S}'$ based on the values of Q and $Temp$, leading to the formation of the offspring population $\mathbb{S}''$.

(1) **Exploration mode ($Q < \Delta_Q$).** When food is limited in the environment, individual snakes enter exploration mode and randomly choose a location to search for food. The update mechanism can be described as Eqs. (19) and (20) [15].

$$X''_{m,y}(l+1) = X'_{m,rand}(l) \pm c_2 \times E_m \times \left(X_{LB} + \tau \times (X_{UB} - X_{LB})\right), \tag{19}$$

$$X''_{f,y}(l+1) = X'_{f,rand}(l) \pm c_2 \times E_f \times \left(X_{LB} + \tau \times (X_{UB} - X_{LB})\right), \tag{20}$$

where $X''_{m,y}$ and $X''_{f,y}$ denote the updated positions of the y-th male individual $X'_{m,y}$ and the y-th female individual $X'_{f,y}$, respectively. $X'_{m,rand}$ and $X'_{f,rand}$ refer to the positions of randomly selected male and female individuals, respectively. c_2 is a constant valued at 0.05, whereas X_{UB} and X_{LB} represent the upper and lower bounds of the solution space, respectively. E_m and E_f signify the exploration abilities of $X'_{m,y}$ and $X'_{f,y}$, respectively, with the corresponding computational expressions provided as follows:

$$E_m = \exp\left(-M(X'_{m,rand})/M(X'_{m,y})\right), \tag{21}$$

$$E_f = \exp\left(-M(X'_{f,rand})/M(X'_{f,y})\right). \tag{22}$$

(2) **Exploitation mode ($Q > \Delta_Q, Temp > \Delta_T$).** In scenarios where food resources are plentiful and environmental temperatures are comparatively high, individual snakes enter exploitation mode, and migration toward the optimal position is favored, as expressed by the following computational expressions [15].

$$X''_{m,y}(l+1) = X^* \pm c_3 \times Temp \times \tau \times \left(X^* - X'_{m,y}(l)\right), \tag{23}$$

$$X''_{f,y}(l+1) = X^* \pm c_3 \times Temp \times \tau \times \left(X^* - X'_{f,y}(l)\right), \tag{24}$$

where X^* denotes the best individual in the parent population $\mathbb{S}'$, and c_3 is a constant that is equal to 2.

(3) **Fight mode ($Q > \Delta_Q, Temp < \Delta_T, \tau > 0.6$).** When food resources are plentiful and environmental temperatures are relatively low, individual snakes exhibit these two distinct behavioral patterns:

- With a probability of 40 %, snakes enter a fight mode, where males compete for the opportunity to mate with the most desirable female, whereas females carefully choose the best male.
- With a probability of 60 %, snakes enter a mating mode, where males and females engage in copulatory activities.

Therefore, in the fight mode, the updating mechanisms for the male individual $X'_{m,y}$ and the female individual $X'_{f,y}$ are defined as Eqs. (25) and (26) [15].

$$X''_{m,y}(l+1) = X'_{m,y}(l) + c_3 \times F_m \times \tau \times \left(Q \times X_f^* - X'_{m,y}(l)\right), \tag{25}$$

$$X''_{f,y}(l+1) = X'_{f,y}(l) + c_3 \times F_f \times \tau \times \left(Q \times X_m^* - X'_{f,y}(l)\right), \tag{26}$$

where X_m^* and X_f^* represent the best male and the best female, respectively, in the parent population $\mathbb{S}'$. F_m and F_f denote the fighting abilities of $X'_{m,y}$ and $X'_{f,y}$, respectively, with the calculations defined as follows:

$$F_m = \exp\left(-M(X_f^*)/M(X'_{m,y})\right), \tag{27}$$

$$F_f = \exp\left(-M(X_m^*)/M(X'_{f,y})\right). \tag{28}$$

(4) **Mating mode ($Q > \Delta_Q, Temp < \Delta_T, \tau < 0.6$).** An interdependent relationship exists between the mating behaviors of each pair of snake

individuals and the availability of food resources. The updating mechanism is outlined as Eqs. (29) and (30) [15].

$$X''_{m,y}(l+1) = X'_{m,y}(l) + c_3 \times M_m \times \tau \times \left(Q \times X'_{f,y}(l) - X'_{m,y}(l)\right), \tag{29}$$

$$X''_{f,y}(l+1) = X'_{f,y}(l) + c_3 \times M_f \times \tau \times \left(Q \times X'_{m,y}(l) - X'_{f,y}(l)\right), \tag{30}$$

where M_m and M_f represent the mating abilities of $X'_{m,y}$ and $X'_{f,y}$, respectively, with the calculations determined by

$$M_m = \exp\left(-M(X'_{f,y})/M(X'_{m,y})\right), \tag{31}$$

$$M_f = \exp\left(-M(X'_{m,y})/M(X'_{f,y})\right). \tag{32}$$

In the mating mode, female individuals may lay eggs that develop into new snake offspring. If the eggs successfully hatch, the worst male individual, X°_m, and the worst female individual, X°_f, are selectively replaced, as indicated by Eqs. (33) and (34) [15].

$$X^{\circ}_m = X_{LB} + \tau \times (X_{UB} - X_{LB}), \tag{33}$$

$$X^{\circ}_f = X_{LB} + \tau \times (X_{UB} - X_{LB}). \tag{34}$$

The dimension value x_i of each snake individual represents the index of the VM assigned to task t_i, which must satisfy the condition $1 \le x_i \le |V|$. Therefore, if $x_i < 1$, then it is adjusted to 1; and if $x_i > |V|$, then it is adjusted to $|V|$.

After the offspring population $\mathbb{S}''$ is obtained, the solution update rules are applied to compare individuals X'_y from the parent population $\mathbb{S}'$ with their corresponding counterparts X''_y in the offspring population $\mathbb{S}''$. Then, the superior individuals X'''_y are chosen to form a new population $\mathbb{S}'''$.

Algorithm 2 Solution update module of AOM-SO.

```
Input: $\mathbb{S}'$, $Q$, $Temp$, $X^*_m$, and $X^*_f$.
Output: $\mathbb{S}''$.
 1: for each $X'_y$ in $\mathbb{S}'$ do
 2:   if $Q < \Delta_Q$ then
 3:     Enter exploration mode and update by using Eqs. (19) and (20);
 4:   else
 5:     if $Temp > \Delta_T$ then
 6:       Enter exploitation mode and update by using Eqs. (23) and (24);
 7:     else
 8:       if $\tau > 0.6$ then
 9:         Enter fight mode and update by using Eqs. (25) and (26);
10:       else
11:         Enter mating mode and update by using Eqs. (29) and (30);
12:       end if
13:     end if
14:   end if
15: end for
16: return $\mathbb{S}''$.
```

4.3. AOM module

The solution update module is derived from the original SO algorithm. It has four distinct update modes, each following specific updating mechanisms. Consequently, solutions may become trapped in local optima during the iterative process. Additionally, the positional changes of the snake individuals within the solution update module arise partly from random exploratory behaviors, whereas the rest are influenced by their movement paths and makespan. This process lacks a clear mechanism focused on cost reduction to optimize their movement strategies, thereby limiting the overall performance and enhancement of the algorithm. An AOM mechanism is proposed to address these shortcomings. The principles of this mechanism are further elucidated through relevant definitions and explanations in the following discussion.

For the population $\mathbb{S}'''$, a number $\delta_{m,rand}$ of male individuals $X'''_{m,rand}$ is randomly chosen from the male group to create the male subgroup R_m. Likewise, a number $\delta_{f,rand}$ of female individuals $X'''_{f,rand}$ is randomly selected from the female group to establish the female subgroup R_f. $\delta_{m,rand}$ and $\delta_{f,rand}$ can be calculated as follows:

$$\delta_{m,rand} = \mu \times \delta_m, \tag{35}$$

$$\delta_{f,rand} = \mu \times \delta_f, \tag{36}$$

where μ is known as the oriented probability, and $\mu = 10\,\%$ is used in this study.

Definition 1: Negative Budget Cost ($\tilde{B}$). If the snake individual X'''_y violates the budget constraint, then the difference between the workflow budget cost $B(G)$ and the total cost $C(X'''_y)$ is negative. In this study, the opposite value of this difference is defined as the negative budget cost $\tilde{B}(X'''_y)$ for the snake individual X'''_y, expressed as follows:

$$\tilde{B}(X'''_y) = -\left(B(G) - C(X'''_y)\right). \tag{37}$$

The negative budget cost indicates the extent to which the total cost of the snake individual X'''_y exceeds the budgeted cost. This study allocates the negative budget cost across various dimensions to reduce the total cost and ensure compliance with the budget constraint. Recognizing that a mapping relationship exists between the individual dimensions and the workflow tasks is essential; thus, this can also be described as distributing the negative budget cost across the relevant tasks.

Each task t_i has a minimum execution cost $C_{\min}(t_i)$ (i.e., the cost required to execute t_i on the least expensive VM). Thus, each task adaptively lowers its execution cost while maintaining $C_{\min}(t_i)$ as a lower limit. This reduction aims to impact the negative budget cost of the snake individual locally, ultimately bringing it down to zero to satisfy the budget constraint.

Definition 2: Adaptiveness (Υ). The adaptiveness Υ_i of task t_i is defined as the difference between the cost needed to execute task t_i on the assigned VM I_k (where k is the integer value of x_i) in the current scheduling scheme and $C_{\min}(t_i)$. It can be computed by

$$\Upsilon_i = C_{exe}(t_i, I_k) - C_{\min}(t_i). \tag{38}$$

Definition 3: Dynamic Sub-budget ($subB$). The usable sub-budget cost $subB(t_i)$ for task t_i is determined by its adaptiveness Υ_i and the negative budget cost $\tilde{B}(X'''_y)$ of the individual. $subB(t_i)$ can be calculated by

$$subB(t_i) = C_{exe}(t_i, I_k) - \tilde{B}(X'''_y) \times \frac{\Upsilon_i}{\sum_{j=1}^{|T|} \Upsilon_j}. \tag{39}$$

The sub-budgets linked to tasks are not static; instead, their values are dynamically modified in real time as the iterative process unfolds, thereby reflecting changes in the individual's total cost. This dynamic sub-budget approach seeks to enhance resource utilization strategies, thereby ensuring flexibility and responsiveness in the algorithm's search process to optimize overall performance. This study incorporates the total number of tasks $|T|$ into the algorithm to enable the proposed algorithm to handle workflow scheduling for tasks of varying scales.

Definition 4: Dimensional Random Sampling. $|D|$ dimensions are randomly selected from X'''_y with an oriented probability μ, denoted as the set $D = \{d_1, \ldots, d_r, \ldots, d_{|D|}\}$. The computational expression for $|D|$ is presented as follows:

$$|D| = \mu \times Dim, \tag{40}$$

where the dimension Dim of X'''_y is equal to the total number of tasks $|T|$ (i.e., the task size). Consequently, the number of dimension random samples $|D|$ changes linearly with the task size, allowing the AOM-SO algorithm to automatically adjust the number of dimension random samples based on the task size of the current workflow.

The randomly selected dimensions in D essentially represent a form of random sampling for workflow tasks, as illustrated in the following examples:

- *Example 1*: Let the total number of tasks be $|T| = 10$. On the basis of Eq. (40), we can derive $|D| = 1$ and $D = \{d_1\}$. Furthermore, the randomly selected dimension is assumed to be the third dimension; in this case, $D = \{3\}$ represents the randomly selected task t_3.
- *Example 2*: Let the total number of tasks be $|T| = 50$. According to Eq. (40), we can determine $|D| = 5$ and $D = \{d_1, d_2, d_3, d_4, d_5\}$. Additionally, if we assume that the randomly chosen dimensions are the 9th, 15th, 18th, 35th, and 47th dimensions, then $D = \{9, 15, 18, 35, 47\}$ represents the randomly selected tasks t_9, t_{15}, t_{18}, t_{35}, and t_{47}.

Definition 5: Oriented Mutation Set (Ω). For the task set represented by D, task t_{d_r} is sequentially selected, and the VM indexes with execution costs that are less than or equal to the sub-budget of task t_{d_r} are identified (i.e., an oriented search for the suitable k is conducted) to form the mutation queue $q(d_r)$ for the d_r-th dimension:

$$q(d_r) = \left\{ k | \exists I_k, C_{exe}(t_{d_r}, I_k) \le subB(t_{d_r}) \right\}. \tag{41}$$

If no I_k that satisfies Eq. (41) exists, then the mutation queue for the d_r-th dimension is defined as follows:

$$q(d_r) = \left\{ k | \exists I_k, C_{exe}(t_{d_r}, I_k) = C_{\min}(t_{d_r}) \right\}. \tag{42}$$

The calculated mutation queues collectively constitute the oriented mutation set Ω for the individual X'''_y across multiple dimensions, represented as

$$\Omega = \left\{ q(d_1), \dots, q(d_r), \dots, q(d_{|D|}) \right\}. \tag{43}$$

Definition 6: AOM Mechanism. The male subgroup R_m and the female subgroup R_f are traversed separately to assess whether each individual $X'''_{m,rand} \in R_m$ or $X'''_{f,rand} \in R_f$ meets the budget constraint. If an individual violates the budget constraint, then its negative budget cost is calculated, and random sampling is performed on the dimensions to obtain the set D. From this set, the oriented mutation set Ω is derived for the individual across multiple dimensions. Each random dimension d_r is traversed in D, where the dimension value is denoted as x_{d_r}. A VM index k is randomly selected from the mutation queue $q(d_r)$ of the oriented mutation set Ω to replace x_{d_r} and continue this process until the traversal is finished.

Base on the aforementioned definition, the AOM mechanism incorporates three random operations to prevent the algorithm from getting trapped in local optima:

- Random selection of the male subgroup R_m and female subgroup R_f. At the population level, individuals are divided into two parts using an oriented probability μ. One group of individuals retains the position features from the solution update module without undergoing oriented mutation to avoid excessive randomness in the population that might hinder algorithm convergence. The other group of individuals (from R_m and R_f) undergoes oriented mutation to enhance diversity in individual positions and prevent the algorithm from being stuck in local optima.
- Random sampling of individual dimensions. At the individual level, certain dimensions are randomly chosen for oriented mutation, with the randomness level controlled by the oriented probability μ. If too many dimensions are randomly selected, the algorithm may fall into new local optima; if too few, it may slow down the search for feasible solutions that meet budget constraints in the early iterations.
- Random mutation of VM indexes. At the task level, one index is randomly selected from the oriented search VM index set for mutation, allowing the AOM-SO algorithm to maintain directionality while incorporating randomness.

Throughout the AOM mechanism, the oriented probability μ represents the collaborative interplay between randomness and directionality. The discussion on the selection of μ values will be further elaborated in Section 5.1.3. Overall, the AOM mechanism balances randomness and orientation. It employs a certain degree of randomness to help escape local optima in the solution update module, thereby increasing search diversification. Additionally, it incorporates orientation, which accelerates the search for feasible solutions that meet budget constraints during the early stages of iteration through multidimensional AOMs. This combination allows for an early transition of the iterative process into the phase focused on minimizing makespan.

The population $\mathbb{S}'''$ is randomly updated using the AOM mechanism to generate a new parent population $\mathbb{S}^{l+1}$ (i.e., $\mathbb{S}'$) for the next iteration. This process continues until the maximum number of iterations is reached.

Algorithm 3 AOM module of AOM-SO.

```
Input: $\mathbb{S}'''$, $\mu$.
Output: $\mathbb{S}^{l+1}$.
 1: Calculate the number of random male individuals $\delta_{m,rand}$;
 2: Calculate the number of random female individuals $\delta_{f,rand}$;
 3: Randomly select $\delta_{m,rand}$ male individuals from $\mathbb{S}'''$;
 4: Randomly select $\delta_{f,rand}$ female individuals from $\mathbb{S}'''$;
 5: for each $X'''_{m,rand}$ in $R_m$ do
 6:   if $C(X'''_{m,rand}) > B(G)$ then
 7:     Calculate $\tilde{B}(X'''_{m,rand})$ by using Eq. (37);
 8:     for $i \leftarrow 1$ to $|T|$ do
 9:       Calculate $\Upsilon_i$ by using Eq. (38);
10:       Calculate $subB(t_i)$ by using Eq. (39);
11:     end for
12:     Obtain a random set of dimensions $D$ for $X'''_{m,rand}$ by using Eq. (40);
13:     for $r \leftarrow 1$ to $|D|$ do
14:       Find $q(d_r)$ for the $d_r$-th dimension;
15:       Add $q(d_r)$ to $\Omega$;
16:       $x_{d_r} \leftarrow$ Randomly select a $k$ from $q(d_r)$;
17:     end for
18:   end if
19: end for
20: Update $R_f$, as per lines 5–19;
21: return $\mathbb{S}^{l+1}$.
```

4.4. A motivational example

To highlight the effectiveness of the AOM-SO algorithm, this study utilizes the workflow diagram presented in Fig. 1 to visually compare the scheduling outcomes of the AOM-SO algorithm with OSAM [14], MG-PRO [8], and the original SO [15] algorithms. The unit prices for VMs I_1, I_2, and I_3 are established at 3, 5, and 7, correspondingly, while the workflow budget is fixed at 650 [42]. The results of the OSAM, MG-PRO, original SO, and AOM-SO algorithms for scheduling this workflow are depicted in Fig. 3. The total execution costs are 649, 650, 650, and 649, with makespans of 87, 98, 88, and 84, respectively. This outcome suggests that the AOM-SO algorithm establishes a more effective task-VM mapping configuration and achieves a shorter makespan when all four algorithms comply with the budget constraints, as illustrated in Fig. 3(d).

4.5. Time complexity analysis

The time complexity of the AOM-SO algorithm is primarily determined by the maximum number of iterations L, the population size δ, and the total number of tasks $|T|$ within the workflow. The initialization module primarily involves the random generation of the initial population, dependent on the population size δ and total number of tasks $|T|$,

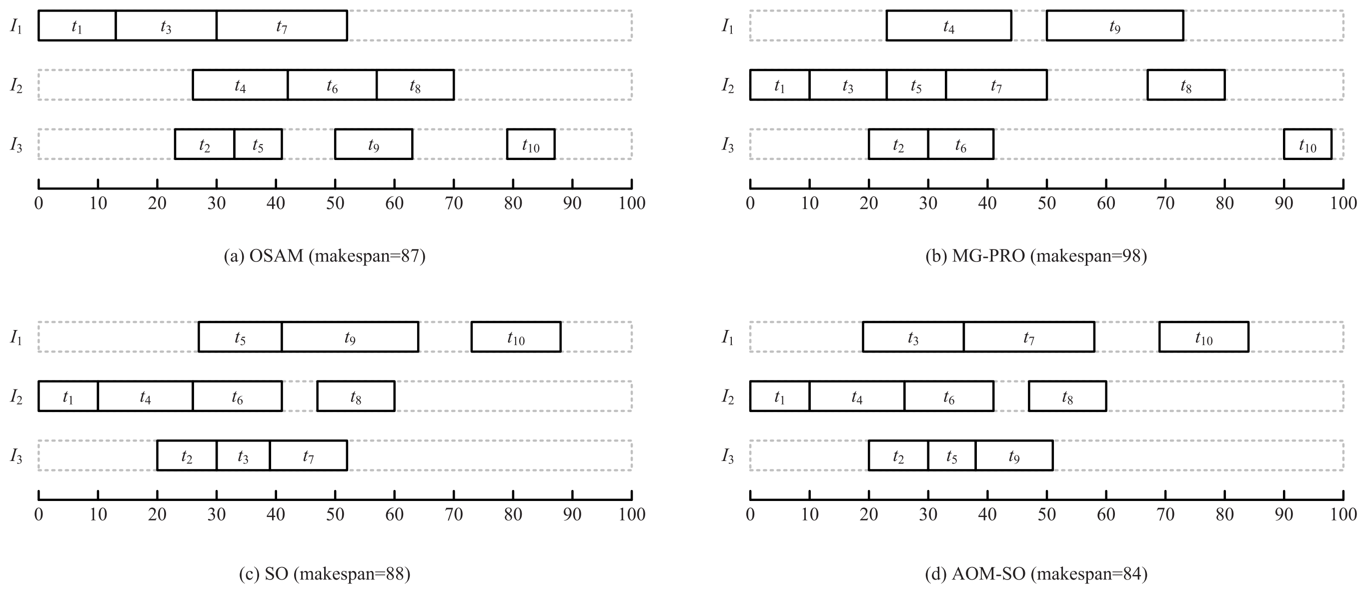


Fig. 3. Schedules of the sample DAG depicted in Fig. 1 with (a) OSAM, (b) MG-PRO, (c) SO, (d) AOM-SO.

with a time complexity of $O(\delta \times |T|)$. The solution update module, as shown in Algorithm 2, operates in each iteration where the population enters only one update mode for updating individual positions, with a time complexity of $O(\delta \times |T|)$. The AOM module, illustrated in Algorithm 3, selects a subset of individuals randomly in each iteration based on the oriented probability μ. For each selected individual, it randomly picks multiple random dimensions using the oriented probability μ for oriented mutation of each selected dimension value. As the total number of tasks in the workflow increases, reflecting higher individual dimensions, and more random dimensions are chosen for oriented mutation in selected individuals, the time complexity is $O(\mu^2 \times \delta \times |T|)$. Since the algorithm undergoes a total of L iterations, the time complexity of the solution update module is $O(L \times \delta \times |T|)$, and that of the AOM module is $O(L \times \mu^2 \times \delta \times |T|)$. Additionally, the time complexity for calculating the cost and makespan of individuals, along with comparing and updating the best individual, is $O(L \times \delta \times |T| + L \times \delta)$. Therefore, the overall time complexity of AOM-SO is $O(\delta \times |T| + L \times \delta \times |T| + L \times \delta)$.

5. Experiments and performance evaluation

This study adopts OSAM [14], MG-PRO [8], SO [15], and AEFT [25] algorithms as benchmarks for comparative assessment. In accordance with the AOM-SO algorithm proposed in this work, all four comparative algorithms are designed to minimize makespan while adhering to budget constraints. A comparative analysis is performed by utilizing five distinct metrics to evaluate the performance of these algorithms comprehensively.

5.1. Experimental setup

The experiments conducted in this study are executed on the CentOS 7.3 operating system, with the algorithm developed and implemented using the Python programming language. Given the challenges associated with conducting reproducible experiments in real data centers or cloud platforms, this study employs simulation experiments to evaluate the performance of the proposed AOM-SO algorithm, as supported by the literature [5,20]. Consistent with prior work [5], our experiments refer to Amazon EC2 to establish nine distinct types of VM instances. Each VM instance type varies in processing speed η_k (measured in MB/s) and pricing p_k (in \$/h), with further details provided in Table 3. Additionally, the average bandwidth bw is set at 20 MB/s.

Table 3
Parameter configuration of 9 VM instance types [5].

Type	η_k	p_k	Type	η_k	p_k
1	1.0	0.12	6	3.5	0.595
2	1.5	0.195	7	4.0	0.72
3	2.0	0.28	8	4.5	0.855
4	2.5	0.375	9	5.0	1.0
5	3.0	0.48	–	–	–

Table 4
Experimental scale settings.

Scale	Dataset	Number of VMs
Small	EP_19, LIGO_40, GE_27, Montage_18	$\|V\| = 3$
Medium	EP_103, LIGO_274, GE_299, Montage_126	$\|V\| = 6$
Large	EP_523, LIGO_544, GE_495, Montage_606	$\|V\| = 12$

5.1.1. Datasets

This study utilizes five types of typical real-world scientific workflow applications [26] as test datasets: **(1)** the epigenomics (EP) workflow application in biogenomics domain, depicted in Fig. 4(a), with task quantities of 19, 51, 103, and 523; **(2)** the LIGO workflow application in the field of gravitational physics, shown in Fig. 4(b), with task quantities of 40, 94, 274, and 544; **(3)** the Gaussian elimination (GE) parallel application, illustrated in Fig. 4(c), with task quantities of 27, 54, 299, and 495; **(4)** the Montage workflow application in astronomy, represented in Fig. 4(d), with task quantities of 18, 66, 126, and 606; and **(5)** molecular dynamics code [43] possesses a fixed DAG structure, demonstrated in Fig. 4(e), encompassing a total of 41 tasks.

This study establishes three datasets of different scales [8,14] to support the experimental validation of the AOM-SO algorithm's effectiveness. The specifications are detailed in Table 4.

5.1.2. Performance metrics

This study employs the following five performance metrics to compare and analyze comprehensively the performance of the five algorithms:

(1) **Success rate.** A critical requirement for minimizing makespan involves ensuring that the scheduling solution adheres to the budget constraint set for a workflow (i.e., the solution obtained is a valid

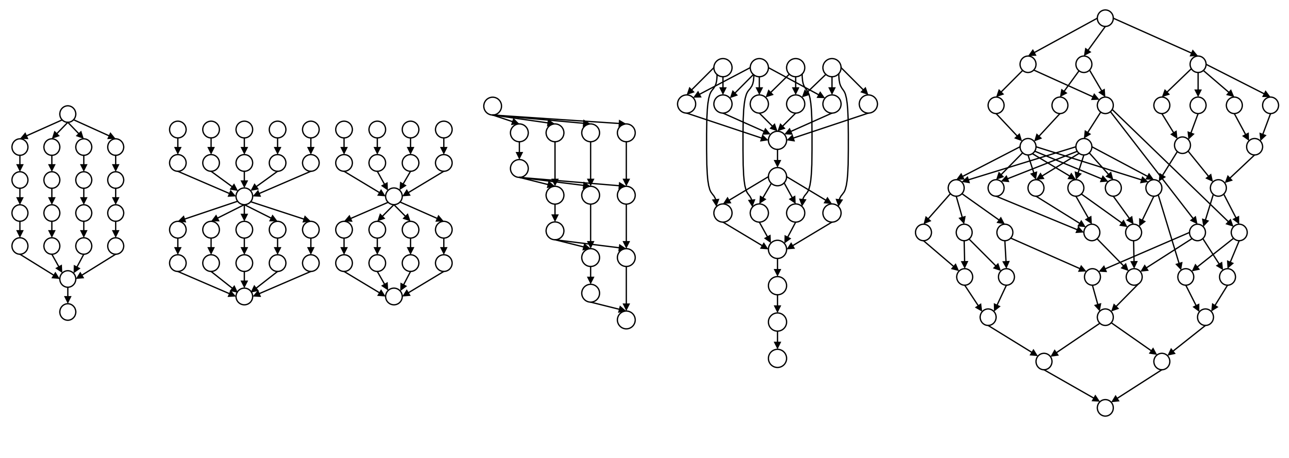


Fig. 4. Five typical scientific workflow applications.

feasible solution). Therefore, the success rate (S_R) is used to evaluate the capability of the five algorithms to find feasible solutions, as follows:

$$S_R = \frac{n}{N}, \tag{44}$$

where n indicates the number of times the algorithm identifies a feasible solution out of N experiments, with N representing the total number of experiments conducted. In this study, N is set to 15, and the evaluation is performed using datasets of small, medium, and large scales as well as molecular dynamics code. A high success rate reflects superior algorithm performance.

(2) **Makespan.** This study designs three sets of comparative experiments that focus on the following aspects to evaluate the makespan performance of the algorithms effectively: different budget constraint conditions, varying task sizes, and diverse configurations of VM quantities.

Experiment 1: Comparison of makespan under varying budget constraints. In this experiment, a molecular dynamics code and a medium-scale dataset are used, incorporating a budget factor β [18] to create varying budget constraints for each workflow. The computational expression for the budget constraint $B(G)$ is as follows:

$$B(G) = \beta \times C_{\min}(G), \tag{45}$$

where $1.0 \le \beta \le 1.4$, with a step size of 0.1. $C_{\min}(G)$ denotes the minimum cost required to execute workflow G and is defined as the total cost incurred when all tasks of the workflow are assigned to the least expensive VMs within the resource pool. The budget constraint begins in a highly constrained state and gradually increases toward a considerably relaxed condition.

Experiment 2: Comparison of makespan under different task sizes. The budget factor ($\beta = 1.4$) and the number of VMs ($|V| = 12$) are held constant to assess the makespan performance of the five algorithms while scheduling four types of workflows (EP, LIGO, GE, and Montage), each with varying task quantities.

Experiment 3: Comparison of makespan with varying VM numbers. In this experiment, the budget factor ($\beta = 1.4$) is kept constant while the makespan of five algorithms is evaluated across five types of workflows. The assessment is carried out using a molecular dynamics code and a medium-scale dataset with the number of VMs set to 6, 12, 18, 24, and 30.

(3) **Relative deviation index (RDI)** [14]**.** This study introduces the relative deviation metric to validate the scheduling advantages of the AOM-SO algorithm, expressed as follows:

$$\mathrm{RDI} = \frac{M - M^*}{M^*}, \tag{46}$$

where M denotes the makespan corresponding to the solution of each algorithm, and M^* represents the best makespan among the results of the five algorithms. The experiments utilize small, medium, and large-scale datasets for testing. Each algorithm is executed independently 15 times for each dataset scale to ensure fairness, with the best result selected for comparison. Then, the corresponding RDI values are calculated, where a small RDI indicates superior algorithm performance.

(4) **Convergence.** The scenario where the algorithm achieves a state of convergence signifies that the iterative evolution process has ceased and an approximate optimal solution has been found. This study records the best solution generated by the algorithm at each iteration on the medium-scale dataset to observe the algorithm's convergence process.

(5) **Runtime.** Runtime serves as a crucial metric for evaluating the efficiency of metaheuristic algorithms in obtaining an approximate optimal solution through extensive iterations. A short runtime indicates a great advantage for the algorithm. In particular, the one with the short runtime is considered relatively superior when two algorithms yield a similar makespan. In this study, the average runtime of the five algorithms is recorded in detail over 15 experiments. The ratio R_{ratio} of the average runtime of each algorithm to the average runtime of the AOM-SO algorithm is calculated and recorded. The expression for this ratio is calculated as follows:

$$R_{ratio} = \frac{R_{algorithm}}{R_{AOM-SO}}, \tag{47}$$

where $R_{algorithm}$ represents the average runtime of each algorithm, while R_{AOM-SO} represents the average runtime of the AOM-SO algorithm. A small ratio signifies a short runtime for the algorithm. The ratio for the AOM-SO algorithm is equal to 1.

5.1.3. Parameter settings

Given that OSAM, MG-PRO, original SO, and AOM-SO algorithms are population-based metaheuristic algorithms, the experiments maintain a consistent population size of $\delta = 50$ and a maximum number of iterations of $L = 1000$. Additionally, the specific parameters for the OSAM, MG-PRO, and original SO algorithms are derived from the original works, as outlined in Table 5.

The AOM-SO algorithm introduces a significant concept known as oriented probability μ, which plays a vital role in balancing randomness and orientation within the AOM mechanism. The experiment evaluates the values of the oriented probability μ in the range of [0, 100] with a step size of 10 using large-scale datasets. As shown in Fig. 5(a), the makespan of the EP, LIGO, GE, and Montage workflows significantly decreases when μ increases from 0 % to 10 %. However, after μ exceeds 10 %, the makespan only fluctuates slightly. To further analyze the impact of μ, the study tests the values of μ in the range of [0, 10] with

Table 5
Parameter settings of different algorithms.

Algorithm	Parameters and values
OSAM [14]	$\beta = 3.0,\ \gamma = 25\,\%$
MG-PRO [8]	$\gamma = 0.08,\ \varphi = 10,\ \zeta = 60$
SO [15]	$c_1 = 0.5,\ c_2 = 0.05,\ c_3 = 2$
AOM-SO	$c_1 = 0.5,\ c_2 = 0.05,\ c_3 = 2,\ \mu = 10\,\%$

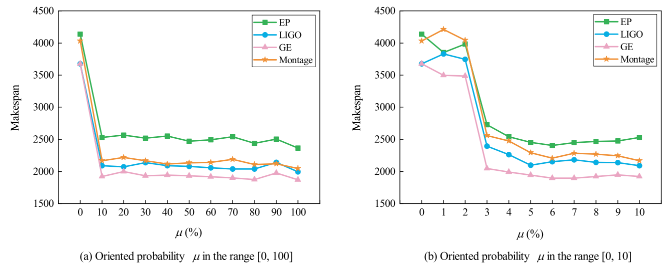


Fig. 5. Numerical tests of the oriented probability μ in the AOM-SO algorithm.

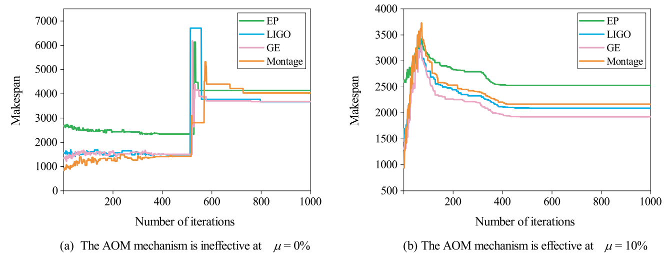


Fig. 6. Utility analysis of the oriented probability μ in the AOM-SO algorithm.

a step size of 1. Fig. 5(b) demonstrates that the significant reduction in makespan for the four types of workflows using the AOM-SO algorithm occurs when μ increases from 2 % to 3 %, stabilizing gradually thereafter. This study ultimately sets the oriented probability μ to 10 % and compares and records the iterative process data of scheduling the EP, LIGO, GE, and Montage workflows under $\mu = 0\,\%$ and $\mu = 10\,\%$ to illustrate the role of $\mu = 10\,\%$ in the AOM-SO algorithm, as shown in Fig. 6.

When $\mu = 0\,\%$, the AOM mechanism is ineffective and does not impact the population's update and iteration process. In this scenario, the AOM-SO algorithm follows the same iterative process as the original SO algorithm, as demonstrated by the convergence characteristics presented in Fig. 6(a). The algorithm identifies only feasible solutions that satisfy the budget constraints after approximately 500 to 600 iterations; at this point, the makespan minimization phase begins. Additionally, the search process is susceptible to being trapped in local optima, which poses certain limitations in optimizing the makespan.

Conversely, when $\mu = 10\,\%$, the AOM mechanism is proved to be effective, leading to a significant change in the population update and iteration process. The convergence characteristics are depicted in Fig. 6(b). After approximately 100 iterations, the algorithm can identify a feasible solution that meets the budget constraint and swiftly moves into the optimization phase to minimize the makespan. In particular, during the iterations from 100 to 500, the algorithm consistently optimizes the makespan until convergence is reached.

In summary, the AOM mechanism significantly improves the convergence speed of the AOM-SO algorithm when the oriented probability is set to $\mu = 10\,\%$; as a result, the issue of the algorithm being trapped in local optima is effectively alleviated. This outcome results in a dual optimization of search efficiency and solution quality.

5.2. Results and analysis

This section offers a comparative analysis of the AOM-SO algorithm's performance in relation to similar algorithms by using the five performance metrics discussed previously.

5.2.1. Budget constraint satisfiability

The comparison results for the success rates of five algorithms across three different scale datasets are presented in Figs. 7–9. Additionally, the results of the success rate comparison on the molecular dynamics code are shown in Fig. 10(a). These figures indicate that the OSAM algorithm performs significantly worse than the four other algorithms. As demonstrated in Fig. 7 and Fig. 10(a), the OSAM algorithm can only identify feasible solutions that adhere to the budget constraints under comparatively lenient budget conditions, even with small-scale datasets and a molecular dynamics code involving 41 tasks. For example, the success rate of the OSAM algorithm in the LIGO workflow depicted in Fig. 7(b) reaches 100 % only when the budget factor is $\beta = 1.4$. Under strict budget conditions, its success rate drops to 0. This finding indicates OSAM's difficulty in finding feasible scheduling solutions under stringent budget constraints. Furthermore, the OSAM algorithm does not reach a success rate of 100 % in medium-scale workflow tests even when the budget constraint is eased to $\beta = 1.4$. In particular, the highest performance is observed in the GE workflow shown in Fig. 8(c), where the success rate reaches only 66.67 %. In the large-scale workflow tests, the success rate remains 0 when $\beta = 1.4$. By contrast, during the tests conducted on the three scale workflows and the molecular dynamics code, the MG-PRO algorithm and the original SO algorithm achieved a 100 % success rate when the budget constraint was relaxed to $\beta = 1.1$. However, under the tightest budget condition of $\beta = 1.0$, the MG-PRO algorithm and the original SO algorithm perform poorly in medium- and large-scale workflow tests, with success rates falling to 0. As a result, finding feasible scheduling solutions is nearly impossible. In comparison, the AOM-SO algorithm maintains a 100 % success rate even under these tight budget constraints while handling large-scale workflows. It is important to note that a 100 % success rate is achieved by the AEFT algorithm, similar to that of the AOM-SO algorithm, for the following reasons: to ensure that adaptability to the scheduling objectives addressed in this study is maintained and to uphold fairness in experimental comparisons, the core strategy focused on minimizing makespan is retained within the AEFT algorithm. Furthermore, the methodology from the work in [17] is adapted to incorporate budget constraint considerations into the AEFT algorithm. This adaptation allows the AEFT algorithm to effectively schedule budget-constrained workflows and generate feasible scheduling solutions.

This study specifically analyzes the success rates of five algorithms at $\beta = 1.0$ to highlight the exceptional performance of the AOM-SO algorithm in searching for feasible solutions that comply with budget constraints. The relevant data are summarized in Table 6. The average success rates indicate that when $\beta = 1.0$, the OSAM algorithm achieves a success rate of 0 %, the MG-PRO algorithm reaches 10.56 %, and the original SO algorithm attains a success rate of 29.44 %. In contrast, both the AEFT algorithm and the AOM-SO algorithm achieve a 100 % success rate. This stark contrast in results underscores the significant advantages offered by the AOM-SO algorithm. Its outstanding performance can largely be attributed to the effectiveness of its AOM mechanism. The multidimensional AOM of random individuals greatly accelerates the search for feasible solutions that comply with budget constraints, thereby facilitating efficient iterations during the makespan minimization phase. Furthermore, the AOM-SO algorithm fully considers the impact of task quantity; thus, it can adapt flexibly to workflow scheduling across various scales.

5.2.2. Makespan evaluation

The results of the comparative analysis of the makespan from Experiment 1 are shown in Figs. 10(b) and 11. The makespan for the OSAM,

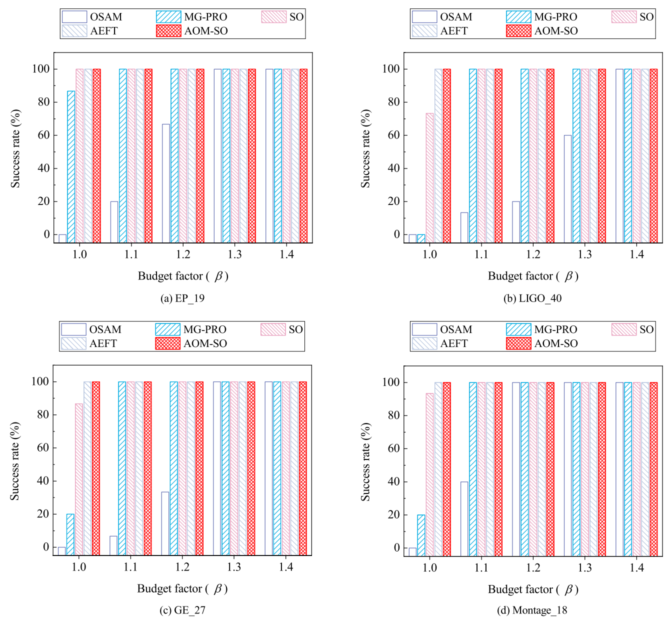


Fig. 7. Success rates of five algorithms on small-scale datasets.

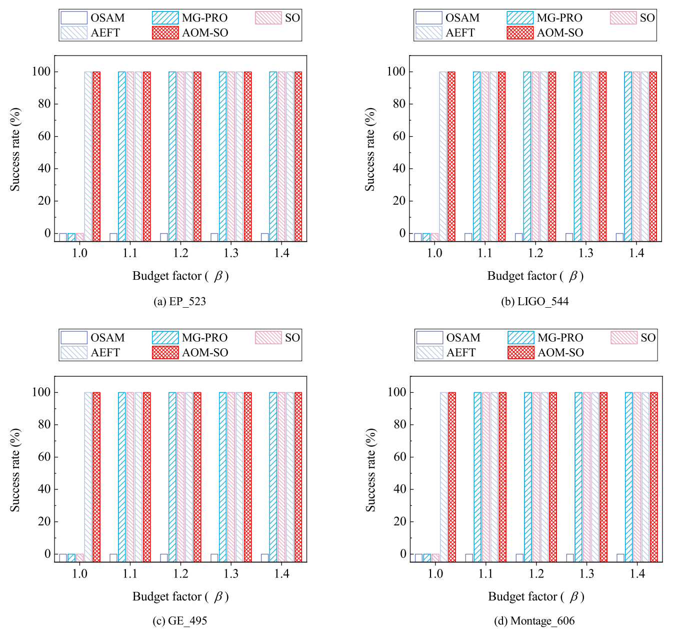


Fig. 9. Success rates of five algorithms on large-scale datasets.

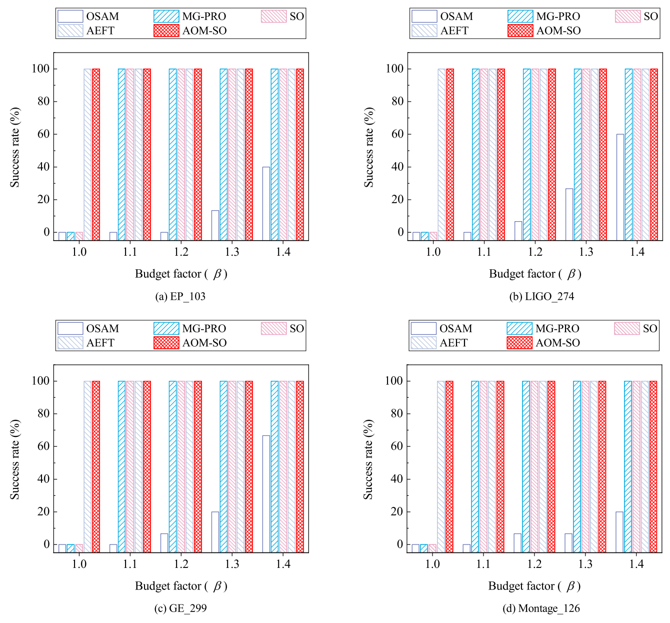


Fig. 8. Success rates of five algorithms on medium-scale datasets.

Table 6
Success rates (%) of the five algorithms when $\beta = 1.0$.

Dataset	Scale	OSAM	MG-PRO	SO	AEFT	AOM-SO
EP	Small	0.00	86.67	100.00	100.00	100.00
	Medium	0.00	0.00	0.00	100.00	100.00
	Large	0.00	0.00	0.00	100.00	100.00
	The average	0.00	28.89	33.33	100.00	100.00
LIGO	Small	0.00	0.00	73.33	100.00	100.00
	Medium	0.00	0.00	0.00	100.00	100.00
	Large	0.00	0.00	0.00	100.00	100.00
	The average	0.00	0.00	24.44	100.00	100.00
GE	Small	0.00	20.00	86.67	100.00	100.00
	Medium	0.00	0.00	0.00	100.00	100.00
	Large	0.00	0.00	0.00	100.00	100.00
	The average	0.00	6.67	28.89	100.00	100.00
Montage	Small	0.00	20.00	93.34	100.00	100.00
	Medium	0.00	0.00	0.00	100.00	100.00
	Large	0.00	0.00	0.00	100.00	100.00
	The average	0.00	6.67	31.11	100.00	100.00
The average		**0.00**	**10.56**	**29.44**	**100.00**	**100.00**

MG-PRO, original SO, AEFT, and AOM-SO algorithms decreases progressively as the budget factor increases. This trend suggests that the algorithms can select appropriate VMs for executing workflow tasks because budget constraints are relaxed. As illustrated in Fig. 8, the OSAM algorithm starts to demonstrate a nonzero success rate for medium-scale dataset tests across the four workflow types of EP, LIGO, GE, and Montage at budget factors of 1.3, 1.2, 1.2, and 1.2, respectively. This finding indicates that the OSAM algorithm can identify only feasible solutions that meet budget constraints and subsequently optimize makespan when these constraints are sufficiently relaxed. Before these thresholds are reached, all iterations of the OSAM algorithm prioritize cost reduction to find feasible solutions that meet budget requirements without focusing on optimizing makespan. Therefore, when analyzing the trends in the makespan changes for the OSAM algorithm across the EP, LIGO, GE, and Montage workflows in Fig. 11, recognizing that the makespan data are valid only when the budget factors reach 1.3, 1.2, 1.2, and 1.2, respectively, is crucial. The same reasoning applies to the MG-PRO and original SO algorithms. As shown in Fig. 11, the AOM-SO algorithm exhibits a significant advantage over the four other algorithms in terms of makespan performance. For instance, Fig. 11(d) depicts that compared with the OSAM, MG-PRO, original SO, and AEFT algorithms, the AOM-SO algorithm for the Montage workflow achieves an average reduction in makespan of 47.33 %, 43.66 %, 48.45 %, and 37.83 %, respectively. However, in the molecular dynamics code shown in Fig. 10(b), with 41 tasks and a simpler search space, the advantage of AOM-SO diminishes compared to its performance on datasets with a larger number of tasks. It shows average reductions in makespan of 17.37 %, 14.39 %, 7.07 %, and 35.31 % when compared to the OSAM, MG-PRO, original SO, and AEFT algorithms, respectively.

The results of the comparative analysis of makespan from Experiment 2 are presented in Fig. 12. As the number of tasks increases, the search space expands, leading to highly complex search paths. As a result, reaching the globally optimal solution becomes increasingly difficult for the algorithms. As illustrated in Fig. 12, the AOM-SO algorithm consistently demonstrates superior performance in makespan across the

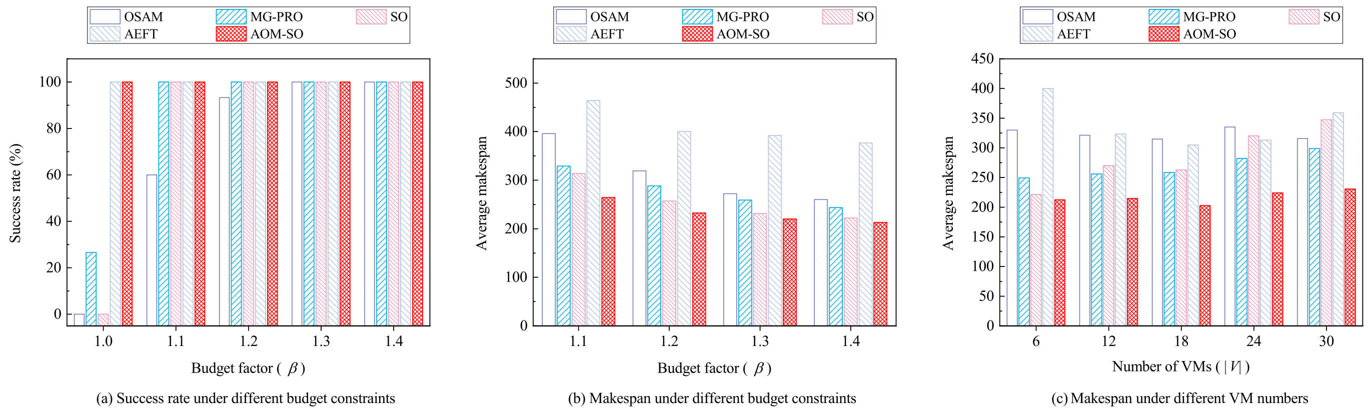


Fig. 10. Performance comparison of five algorithms on molecular dynamics code.

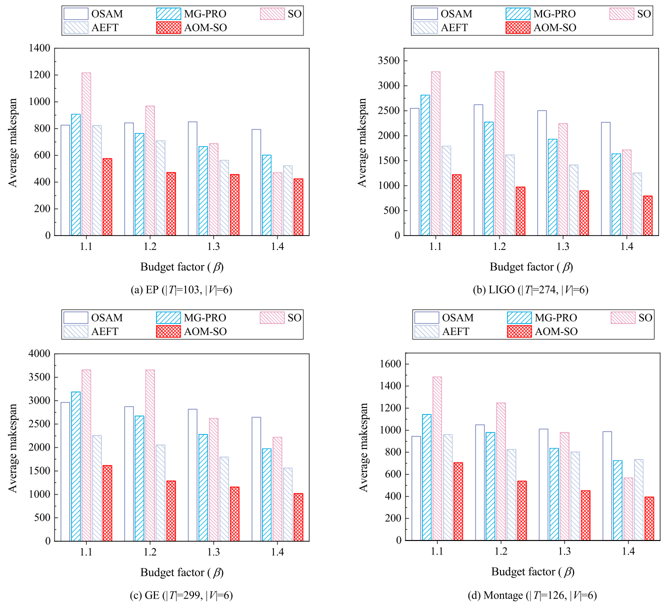


Fig. 11. Comparative results of makespan under different budget constraints.

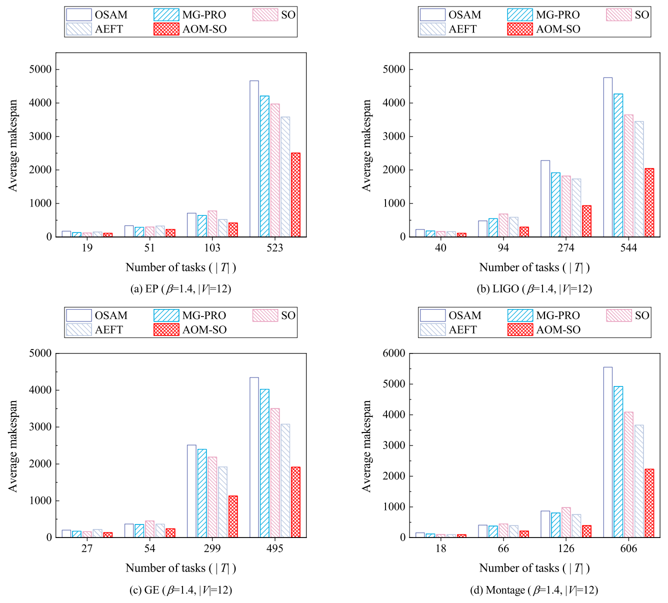


Fig. 12. Comparative results of makespan under different task sizes.

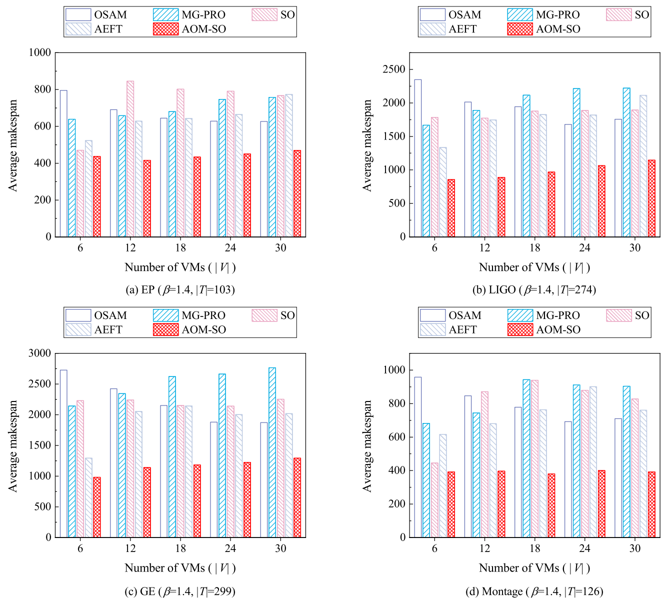


Fig. 13. Comparative results of makespan under different VM numbers.

four types of workflow tests, regardless of the number of tasks. The average values obtained from the four sets of experimental results indicate that compared with the OSAM, MG-PRO, original SO, and AEFT algorithms, the AOM-SO algorithm achieves an average reduction in makespan of 46.11 %, 39.11 %, 38.47 %, and 34.46 %, respectively.

The results of the comparative analysis of the makespan from Experiment 3 are displayed in Figs. 10(c) and 13. Under varying conditions regarding the number of VMs, the proposed AOM-SO algorithm demonstrates superior performance in reducing the execution time of workflows. For instance, compared with the four other algorithms, the AOM-SO algorithm in the LIGO workflow shown in Fig. 13(b) achieves an average reduction in makespan of 48.21 %, 51.27 %, 46.72 %, and 43.88 %.

The comprehensive evaluation reveals that the AOM mechanism designed in this study optimizes the scheduling strategy. Thus, its adaptability to workflows with varying task quantities is demonstrated, and highly efficient VM allocation is facilitated. This optimization results in a reduction in makespan and improves the overall performance of the algorithm. Compared with the OSAM, MG-PRO, original SO, and AEFT algorithms, the AOM-SO algorithm achieves average reductions of 47.32 %, 44.33 %, 44.67 %, and 35.83 %, respectively, in the makespan comparison experiments. These findings underscore the significant ad-

Table 7
RDI values (%) of the five algorithms when $\beta = 1.4$.

Scale	Dataset	OSAM	MG-PRO	SO	AEFT	AOM-SO
Small	EP_19	16.10	**0**	5.93	4.24	3.39
	LIGO_40	34.97	7.69	0.70	2.10	**0**
	GE_27	21.99	7.09	9.22	**0**	8.51
	Montage_18	22.33	1.94	4.85	**0**	3.88
	The average	23.85	4.18	5.18	1.58	3.95
Medium	EP_103	65.44	38.97	7.11	28.19	**0**
	LIGO_274	178.83	78.56	21.30	66.98	**0**
	GE_299	116.46	84.27	33.95	61.49	**0**
	Montage_126	171.75	79.10	16.67	107.34	**0**
	The average	133.12	70.22	19.76	66.00	**0**
Large	EP_523	NA	49.05	54.85	45.03	**0**
	LIGO_544	NA	92.03	71.13	73.30	**0**
	GE_495	NA	92.70	76.48	69.92	**0**
	Montage_606	NA	114.44	86.89	70.40	**0**
	The average	NA	87.06	72.34	64.66	**0**
The average		78.49	53.82	32.43	44.08	1.32

vantages of the AOM-SO algorithm and provide strong validation of its effectiveness in minimizing workflow makespan.

5.2.3. *Relative performance evaluation*

The RDI values of the OSAM, MG-PRO, original SO, AEFT, and AOM-SO algorithms at $\beta = 1.4$ are shown in Table 7. This table provides a quantitative analysis of the performance of the five algorithms in minimizing workflow makespan. An RDI value of 0 for an algorithm indicates that this algorithm produces the shortest workflow makespan among the others and performs optimally on this metric. As illustrated in Fig. 9, the OSAM algorithm shows a success rate of 0 on large-scale datasets at $\beta = 1.4$; thus, the RDI values for this dataset are not included in the table. The average values from Table 7 demonstrate that the AOM-SO algorithm consistently outperforms the others across small, medium, and large datasets. In tests with small-scale datasets, it is notable that AOM-SO does not achieve an RDI value of 0 in the EP, GE, and Montage workflows. Despite this, it still surpasses the OSAM and original SO algorithms. However, the margin of superiority over the AEFT algorithm diminishes. This outcome can be attributed to the modest number of tasks within the small-scale workflow, resulting in a relatively simple and restricted search space. Consequently, the full potential advantages of the AOM-SO algorithm are not entirely showcased.

Overall, the average RDI values for the OSAM, MG-PRO, original SO, AEFT, and AOM-SO algorithms are 78.49 %, 53.82 %, 32.43 %, 44.08 %, and 1.32 %, respectively. This comparison underscores the significant advantage of the AOM-SO algorithm in minimizing makespan. This finding places the AOM-SO algorithm significantly ahead of the four other algorithms. Furthermore, it validates that the AOM mechanism effectively overcomes the limitations of the original SO algorithm, which is prone to falling into local optima. The AOM-SO algorithm consistently achieves optimal makespan by efficiently utilizing computational resources within a constrained search budget.

5.2.4. *Convergence evaluation*

Fig. 14 depicts the changes in the makespan generated by each algorithm as the number of iterations increases. This study showcases the convergence processes of the five algorithms across four classes of medium-scale workflows, particularly under a budget factor of $\beta = 1.4$. This choice is made because, under these conditions, all five algorithms can produce feasible solutions, and the moderate number of tasks demonstrate the algorithms' evolution and transitions throughout the iterative process. Given that this study aims to optimize the workflow makespan under budget constraints, cost and makespan represent conflicting objective variables. Consequently, in the early stages of iteration, the OSAM, MG-PRO, original SO, and AOM-SO algorithms prioritize cost reduction to find feasible solutions that satisfy the budget constraints

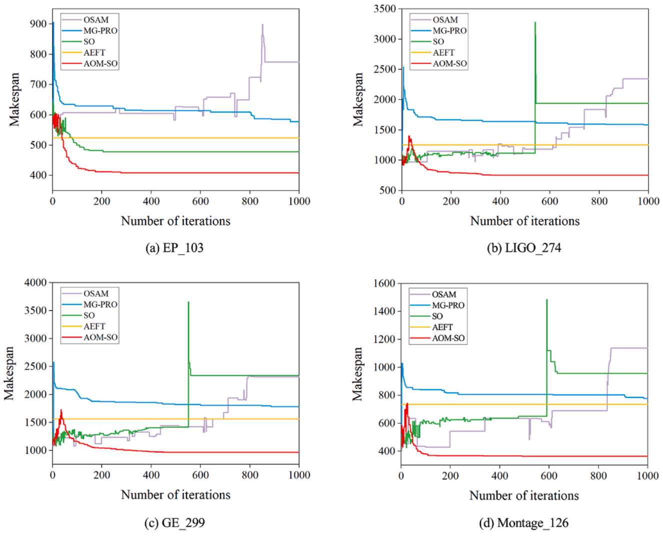


Fig. 14. Comparison of iterative processes for five algorithms.

before they shift their focus to optimizing makespan. As a result, the makespan of these algorithms may experience significant fluctuations in the initial iterations and may even increase at times, leading to convergence curves that do not consistently decrease with each iteration. Additionally, because the AEFT algorithm is a heuristic scheduling algorithm, the scheduling scheme generated by each iteration remains consistent when the workflow data is fixed. This characteristic results in the AEFT algorithm producing a straight line over 1000 iterations, as illustrated in Fig. 14.

The OSAM algorithm exhibits prolonged fluctuations in its convergence curve during the phase of identifying feasible solutions that satisfy budget constraints, with the GE workflow illustrated in Fig. 14(c) taken as an example. The overall tendency indicates a susceptibility to being trapped in local optima, which hinders its effectiveness in minimizing the makespan. By contrast, the MG-PRO algorithm can rapidly identify feasible solutions that satisfy budget constraints in the early stages of iteration. Thus, it can shift focus toward minimizing the makespan. However, this rapid identification comes with the risk of overlooking opportunities to achieve a globally optimal solution. A detailed comparison of the convergence characteristics between the AOM-SO algorithm and the original SO algorithm is presented in Section 5.1.3. The AOM-SO algorithm effectively balances randomness and directionality. Thus, the convergence is accelerated, and the likelihood of falling into local optima is reduced. Once the AOM-SO algorithm successfully identifies a feasible solution that adheres to budget constraints, it continuously optimizes the makespan until convergence is achieved. Compared with the OSAM, MG-PRO, original SO, and AEFT algorithms, the AOM-SO algorithm demonstrates superior performance by finding appropriate solutions and achieving short makespan.

5.2.5. *Runtime evaluation*

This study assesses the efficiency of the five algorithms by using the workflow task sets of small, medium, and large scales. The runtime ratio results are presented in Fig. 15. However, the AEFT algorithm is a heuristic method that efficiently generates scheduling plans by establishing scheduling rules without the need for multiple iterations of optimization. Compared to meta-heuristic approaches, the AEFT algorithm offers a significant advantage in terms of runtime. To ensure fairness in the runtime comparison, this study avoids directly contrasting the runtime of the AEFT algorithm with that of the other four meta-heuristics (OSAM, MG-PRO, original SO, and AOM-SO). In the small-scale dataset, the OSAM algorithm demonstrates a comparatively shorter runtime due to its relatively straightforward update formula. However, as the task

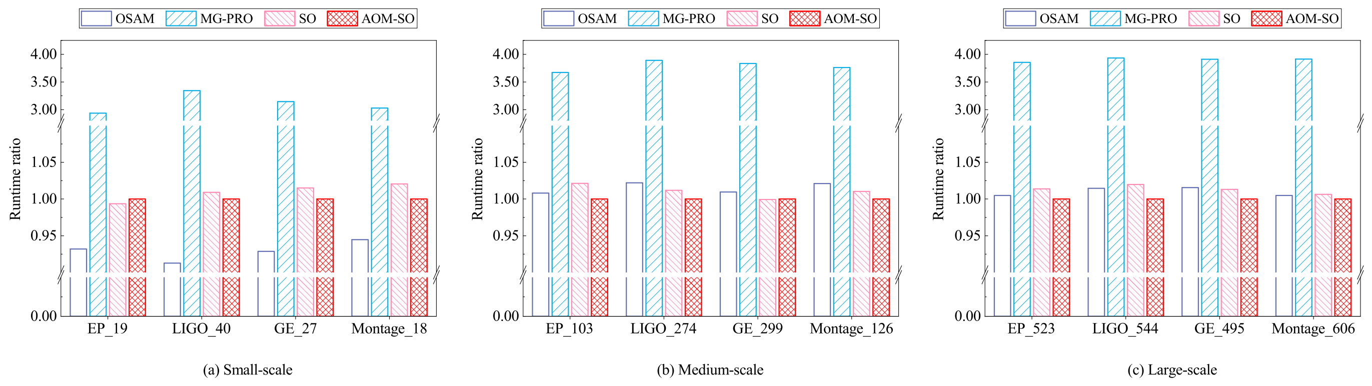


Fig. 15. Comparison of runtimes for four algorithms at different workflow scales.

scale increases and the search space becomes increasingly complex, the runtime ratio of the OSAM algorithm does not show a significant advantage over the runtime ratios of the original SO algorithm and the AOM-SO algorithm, with the runtimes for all three algorithms being comparable. The MG-PRO algorithm incurs the longest execution time across all three datasets because of the complexity of the crossover and mutation operations involved in its evolutionary process. By contrast, the proposed AOM-SO algorithm consistently maintains a runtime close to that of the original SO algorithm and slightly outperforms the OSAM algorithm and the original SO algorithm in the large-scale dataset. This finding demonstrates its superiority over the MG-PRO algorithm across all three dataset sizes.

In summary, the MG-PRO algorithm does not exhibit any corresponding improvement in solution quality even though its runtime is approximately three times that of the AOM-SO algorithm. By contrast, the AOM-SO algorithm demonstrates significant advantages in solution quality under similar runtime conditions when compared with the OSAM algorithm and the original SO algorithm. This benefit is strongly supported by the experimental results of the previously discussed performance metrics. The outstanding performance of the AOM-SO algorithm can largely be attributed to the AOM mechanism, which effectively enables a dual optimization of search efficiency and solution quality, thereby ensuring effectiveness and competitiveness in addressing scheduling challenges.

6. Conclusion and future work

This study introduces the AOM-SO algorithm to address the budget-constrained workflow scheduling problem in heterogeneous cloud environments. Its primary goal is to minimize the makespan of workflows. By incorporating an AOM mechanism that balances randomness and directionality, the AOM-SO algorithm effectively prevents getting stuck in local optima and enhances the search for feasible solutions. We conduct multiple comparative experiments on scientific workflows to assess the performance of the AOM-SO algorithm. The results reveal that the AOM-SO algorithm achieves a 100 % success rate in identifying feasible solutions that meet budget constraints. It outperforms the OSAM, MG-PRO, original SO, and AEFT algorithms. Additionally, it reduces makespan by an average of 43.03 %, showcasing significant advantages in scheduling efficiency. However, the oriented probability μ devised in the AOM-SO algorithm is currently set to 10 % following tests on large-scale datasets and remains unchanged when scheduling workflows with varying structures. While this simplifies the algorithm implementation, it hampers the algorithm's adaptability to diverse workflow structures, potentially impacting the scheduling stability of the AOM-SO algorithm in small-scale workflows.

Future research will explore dynamic μ-tuning and online learning to address current limitations and focus on developing dynamic workflow scheduling algorithms that incorporate uncertainty-aware mechanisms for task execution times. This adaptation is essential for effectively addressing the complexities inherent in real-world scheduling scenarios.

CRediT authorship contribution statement

Yanfen Zhang: Writing – review & editing, Writing – original draft, Methodology; **Longxin Zhang:** Writing – review & editing, Writing – original draft, Funding acquisition, Formal analysis; **Buqing Cao:** Methodology, Data curation; **Jing Liu:** Software, Methodology; **Wenyu Zhao:** Software, Data curation; **Jianguo Chen:** Validation, Methodology; **Keqin Li:** Supervision, Methodology.

Data availability

The source code that support the findings of this study are available in/from hyperlink to data source https://github.com/Yanfen-Zhang/AOM-SO.

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

The authors would like to thank two anonymous reviewers for their suggestions to improve the manuscript. The preliminary version of this paper has been accepted for presentation at the 2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA). This work was partially funded by the Postgraduate Scientific Research Innovation Project of Hunan Province (Grant No. CX20240916), the Scientific research and innovation Foundation of Hunan University of Technology (Grant No. LXBZZ2416), the Natural Science Foundation of Hunan Province, China (Grant Nos. 2023JJ50204, 2024JJ7154), the Scientific Research Foundation of Hunan Provincial Education Department, China (Grant Nos. 23B0560, 24C0291), the National Key R&D Program of China (Grant No. 2018YFB1003401), the National Natural Science Foundation of China (Grant Nos. 61702178, 62072172, 62572186, 62002110, 62372486), CCF C Sangfor 'Yuanwang' Research Fund (Grant No. CCF-SANGFOR OF 2024002), the Open Project Funding of the Key Laboratory of Intelligent Sensing System and Security (Hubei University), Ministry of Education, and the State Key Laboratory Program for Novel Software Technology (Grant No. KFKT2024B52).

Supplementary material

Supplementary material associated with this article can be found in the online version at 10.1016/j.future.2025.108118.

References

[1] G. Zhou, W. Tian, R. Buyya, R. Xue, L. Song, Deep reinforcement learning-based methods for resource scheduling in cloud computing: a review and future directions, Artif. Intell. Rev. 57 (124) (2024). https://doi.org/10.1007/s10462-024-10756-9
[2] L. Zhang, R. Tan, B. Cao, L.A.K. Li, K. Li, EP-MUSTO: entropy-enhanced DRL-based task offloading in secure multi-UAV-assisted collaborative edge computing, IEEE Internet Things J. 12 (16) (2025) 34459–34470. https://doi.org/10.1109/JIOT.2025.3577487
[3] Z. Tong, X. Deng, H. Chen, J. Mei, DDMTS: a novel dynamic load balancing scheduling scheme under SLA constraints in cloud computing, J Parallel Distrib. Comput. 149 (2021) 138–148. https://doi.org/10.1016/j.jpdc.2020.11.007
[4] L. Zhang, M. Ai, K. Liu, J. Chen, K. Li, Reliability enhancement strategies for workflow scheduling under energy consumption constraints in clouds, IEEE Trans. Sustain. Comput. 9 (2) (2024) 155–169. https://doi.org/10.1109/TSUSC.2023.3314759
[5] L. Ye, L. Yang, Y. Xia, X. Zhao, A cost-driven intelligence scheduling approach for deadline-constrained IoT workflow applications in cloud computing, IEEE Internet Things J. 11 (9) (2024) 16033–16047. https://doi.org/10.1109/JIOT.2024.3351630
[6] X. Tang, F. Liu, B. Wang, D. Xu, J. Jiang, Q. Wu, C.L.P. Chen, Workflow scheduling based on asynchronous advantage actor-critic algorithm in multi-cloud environment, Expert Syst. Appl. 258 (2024) 125245. https://doi.org/10.1016/j.eswa.2024.125245
[7] S. Abdi, M. Ashjaei, S. Mubeen, Deadline-constrained security-aware workflow scheduling in hybrid cloud architecture, Future Gener. Comput. Syst. 162 (2025) 107466. https://doi.org/10.1016/j.future.2024.07.044
[8] H. Li, B. Chen, J. Huang, J.R.C. Abreu, S. Chai, Y. Xia, Mutation-driven and population grouping PRO algorithm for scheduling budget-constrained workflows in the cloud, Cluster Comput. 27 (2024) 1137–1158. https://doi.org/10.1007/s10586-023-04006-w
[9] M. Fan, X. Zhao, X. Zuo, L. Ye, A budget-constrained workflow scheduling approach with priority adjustment and critical task optimizing in clouds, IEEE Trans. Autom. Sci. Eng. 22 (2025) 6907–6921. https://doi.org/10.1109/TASE.2024.3456762
[10] J. Zhang, X. Li, L. Chen, R. Ruiz, Scheduling workflows with limited budget to cloud server and serverless resources, IEEE Trans. Serv. Comput. 17 (4) (2024) 1766–1779. https://doi.org/10.1109/TSC.2023.3332697
[11] 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. https://doi.org/10.1016/j.future.2022.06.012
[12] 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. https://doi.org/10.1016/j.eswa.2023.120972
[13] H. Li, G. Xu, B. Chen, S. Huang, Y. Xia, S. Chai, Dual-mutation mechanism-driven snake optimizer for scheduling multiple budget constrained workflows in the cloud, Appl. Soft. Comput. 149 (2023) 110966. https://doi.org/10.1016/j.asoc.2023.110966
[14] H. Li, D. Wang, G. Xu, Y. Yuan, Y. Xia, Improved swarm search algorithm for scheduling budget-constrained workflows in the cloud, Soft Comput. 26 (2022) 3809–3824. https://doi.org/10.1007/s00500-022-06782-w
[15] F.A. Hashim, A.G. Hussien, Snake optimizer: a novel meta-heuristic optimization algorithm, Knowl. Based Syst. 242 (2022) 108320. https://doi.org/10.1016/j.knosys.2022.108320
[16] L. Zhang, Y. Zhang, X. Lu, R. Tan, X. Huang, J. Chen, Budget-aware Scheduling Algorithm Using Negative Offset Mechanism for Snake Optimization in Heterogeneous Cloud, in: 2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2024, pp. 302–309. https://doi.org/10.1109/ISPA63168.2024.00046
[17] H. Arabnejad, J.G. Barbosa, A budget constrained scheduling algorithm for workflow applications, J. Grid Comput. 12 (2014) 665–679. https://doi.org/10.1007/s10723-014-9294-7
[18] K.K. Chakravarthi, L. Shyamala, V. Vaidehi, Budget aware scheduling algorithm for workflow applications in IaaS clouds, Cluster Comput. 23 (2020) 3405–3419. https://doi.org/10.1007/s10586-020-03095-1
[19] S. Kushwaha, R.S. Singh, Deadline and budget-constrained archimedes optimization algorithm for workflow scheduling in cloud, Cluster Comput 28 (117) (2025). https://doi.org/10.1007/s10586-024-04702-1
[20] 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) (2024) 1564–1579. https://doi.org/10.1109/TASE.2023.3247973
[21] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274. https://doi.org/10.1109/71.993206
[22] H. Djigal, J. Feng, J. Lu, J. Ge, IPPTS: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1057–1071. https://doi.org/10.1109/TPDS.2020.3041829
[23] S. Qin, D. Pi, Z. Shao, AILS: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment, Expert Syst. Appl. 198 (2022) 116824. https://doi.org/10.1016/j.eswa.2022.116824
[24] H.R. Faragardi, M.R. Saleh Sedghpour, S. Fazliahmadi, T. Fahringer, N. Rasouli, GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds, IEEE Trans. Parallel Distrib. Syst. 31 (6) (2020) 1239–1254. https://doi.org/10.1109/TPDS.2019.2961098
[25] M. Wang, H. Wang, S. Qiao, J. Chen, Q. Xie, C. Guo, Heterogeneous system list scheduling algorithm based on improved optimistic cost matrix, Future Gener. Comput. Syst. 164 (2025) 107576. https://doi.org/10.1016/j.future.2024.107576
[26] L. Zhang, M. Ai, R. Tan, J. Man, X. Deng, K. Li, Efficient prediction of makespan matrix workflow scheduling algorithm for heterogeneous cloud environments, J. Grid Comput. 21 (75) (2023). https://doi.org/10.1007/s10723-023-09711-9
[27] J. Jiang, Z. Sun, R. Lu, L. Pan, Z. Peng, Real relative encoding genetic algorithm for workflow scheduling in heterogeneous distributed computing systems, IEEE Trans. Parallel Distrib. Syst. 36 (1) (2025) 1–14. https://doi.org/10.1109/TPDS.2024.3492210
[28] H. Li, L. Tian, G. Xu, J.R.C. Abreu, S. Huang, S. Chai, Y. Xia, Co-evolutionary and elite learning-based bi-objective poor and rich optimization algorithm for scheduling multiple workflows in the cloud, Future Gener. Comput. Syst. 152 (2024) 99–111. https://doi.org/10.1016/j.future.2023.10.015
[29] Z.-G. Chen, Z.-H. Zhan, Y. Lin, Y.-J. Gong, T.-L. Gu, F. Zhao, H.-Q. Yuan, X. Chen, Q. Li, J. Zhang, Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach, IEEE Trans. Cybern. 49 (8) (2019) 2912–2926. https://doi.org/10.1109/TCYB.2018.2832640
[30] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, K. Li, Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems, Future Gener. Comput. Syst. 74 (2017) 1–11. https://doi.org/10.1016/j.future.2017.03.008
[31] D. Wu, X. Wang, X. Wang, R. Zeng, M. Huang, Differentially private and truthful auction-based resource procurement for budget-constrained DAG applications in clouds, Comput. Netw. 251 (2024) 110628. https://doi.org/10.1016/j.comnet.2024.110628
[32] Y. Hao, C. Zhao, Z. Li, B. Si, H. Unger, A learning and evolution-based intelligence algorithm for multi-objective heterogeneous cloud scheduling optimization, Knowl. Based Syst. 286 (2024) 111366. https://doi.org/10.1016/j.knosys.2024.111366
[33] S. Rathi, R. Nagpal, G. Srivastava, D. Mehrotra, A multi-objective fitness dependent optimizer for workflow scheduling, Appl. Soft Comput. 152 (2024) 111247. https://doi.org/10.1016/j.asoc.2024.111247
[34] Y. Guo, B. Liu, W. Lin, L.Y. Pan, J.Z. Wang, Dynamic neighborhood grouping-based multi-objective scheduling algorithm for workflow in hybrid cloud, Future Gener. Comput. Syst. 166 (2025) 107633. https://doi.org/10.1016/j.future.2024.107633
[35] P.V. Reddy, K.G. Reddy, An energy efficient RL based workflow scheduling in cloud computing, Expert Syst. Appl. 234 (2023) 121038. https://doi.org/10.1016/j.eswa.2023.121038
[36] N. Rizvi, D. Ramesh, L. Wang, A. Basava, A workflow scheduling approach with modified fuzzy adaptive genetic algorithm in iaas clouds, IEEE Trans. Serv. Comput. 16 (2) (2023) 872–885. https://doi.org/10.1109/TSC.2022.3174112
[37] X. Cai, Y. Zhang, M. Li, L. Wu, W. Zhang, J. Chen, Dynamic deadline constrained multi-objective workflow scheduling in multi-cloud environments, Expert Syst. Appl. 258 (2024) 125168. https://doi.org/10.1016/j.eswa.2024.125168
[38] N. Rizvi, R. Dharavath, D.R. Edla, Cost and makespan aware workflow scheduling in iaas clouds using hybrid spider monkey optimization, Simul. Modell. Pract. Theory 110 (2021) 102328. https://doi.org/10.1016/j.simpat.2021.102328
[39] J. Pan, Y. Wei, A deep reinforcement learning-based scheduling framework for real-time workflows in the cloud environment, Expert Syst. Appl. 255 (2024) 124845.
[40] B. Sun, M. Theile, Z. Qin, D. Bernardini, D. Roy, A. Bastoni, M. Caccamo, Edge generation scheduling for DAG tasks using deep reinforcement learning, IEEE Trans. Comput. 73 (4) (2024) 1034–1047. https://doi.org/10.1109/TC.2024.3350243
[41] M. Fan, L. Ye, X. Zuo, X. Zhao, A bidirectional workflow scheduling approach with feedback mechanism in clouds, Expert Syst. Appl. 249 (2024) 123494. https://doi.org/10.1016/j.eswa.2024.123494
[42] N. Rizvi, D. Ramesh, Fair budget constrained workflow scheduling approach for heterogeneous clouds, Cluster Comput. 23 (2020) 3185–3201. https://doi.org/10.1007/s10586-020-03079-1
[43] 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) (2023) 1807–1821. https://doi.org/10.1109/TSC.2022.3196620