Mutation-based Group Teaching Optimization Algorithm for Scheduling Cloud Workflows with Deadline Constraints

Jiwei Meng[1], Guanghao Xu[1], Bing Chen[1], Huifang Li[*1], Lingguo Cui[1], Senchun Chai[1]

[1] School of Automation, Beijing Institute of Technology, Beijing 100081, P. R. China.
Email: mengjiweibella@icloud.com; 3220200812@bit.edu.cn; chenbing0fff@163.com; huifang@bit.edu.cn; cuilingguo@bit.edu.cn; chaisc97@163.com

Abstract. Cloud computing shows great potential in dealing with immense amounts of data, which satisfies users' demand for high performance computing capability. The elasticity of the cloud and the heterogeneity of virtual machines provisioned provide a cost-effective deploying environment for executing user applications. How to manage and schedule these applications in order to guarantee users' QoS requirements and improve resource utilization, is still a great challenge. In this paper, we design a Group Teaching Optimization Algorithm (GTOA)-based workflow scheduler to minimize the workflow execution cost under deadline constraints. To improve the global search ability and avoid falling into local optima, we introduce a mutation strategy into the original GTOA and propose a Mutation-based GTOA, i.e., M-GTOA. A set of experiments are conducted to verify its effectiveness and experimental results show that M-GTOA outperforms GTOA in success rate and cost optimization. Compared with GA, PSO and OSAM, M-GTOA achieves better optimization performance in most cases, especially for Montage and Sipht workflows.

Keywords: Group teaching optimization algorithm, Workflow scheduling, Cloud computing, Meta-heuristic algorithms

1. INTRODUCTION

Today, the continuously expansion of data brings great demand of a higher computation ability. In response to this trend, a new paradigm called cloud computing is emerged, which makes high performance computing can be realized via cloud resources, i.e., virtual machines (VMs). It delivers flexible and elastic computing services to users in a pay-as-you-go model through Internet [1], and people can rent VMs anytime and anywhere. Workflow scheduling refers to assigning workflow tasks to a set of available VMs to satisfy the requirements of user Quality of Service (QoS). Especially for large-scale scientific workflows, specific scheduling schemes are required to generate a solution to improve resource utilization.

Recently, workflow scheduling is a hot topic and attractive, thus many approaches have been proposed accordingly, which can be divided mainly into two categories, the heuristics and meta-heuristics. Heuristics like the Heterogeneous Earliest Finish Time (HEFT) [2] and Critical Path on a Processor (CPOP) [2] are widely used in scheduling workflows. Y. Yuan et al. propose DET (Deadline Early Tree) [3] aiming to optimize the cost of workflow execution within its deadline. However, the application of heuristics is limited since they cannot guarantee the optimality of obtained solutions. In contrast, meta-heuristics are random or iterative search techniques, and show great potential in solving discrete optimization problems, thus have been widely used in workflow scheduling.

As typical meta-heuristics, Genetic Algorithm (GA) [4], Particle Swarm Optimization (PSO) [5] and Ant Colony Optimization (ACO) [6] are extensively studied and used for scheduling workflows in the cloud. Novel algorithms, e.g., the Owl Search Algorithm with Mutation strategies (OSAM) [7], is also applied into these problems. However, most of the meta-heuristic algorithms like GA and PSO, have special parameters which need to be tuned for different problems. That is inconvenient for use. While other meta-heuristic algorithms like Teaching-Learning-Based Optimization (TLBO) [8] and Neural Network Algorithm (NNA) [9] that only have common parameters, i.e., the size of population and the maximum number of iterations. However, TLBO converges fast and easily falls into local optima.

Group Teaching Optimization Algorithm (GTOA) [10] is a new meta-heuristic algorithm, coming from the real-world learning process. A student can learn from the teacher, other students and himself. As a newly proposed meta-heuristic, GTOA does not have special parameters to set, thus has advantages in use flexibility and convenience over other peers. GTOA's convergence speed is relatively moderate and it is a promising approach for scheduling workflows. In this paper, we propose a Mutation-based GTOA (M-GTOA) by embedding a mutation operation at the end of the student phase in GTOA so as to improve optimization performance. Benefiting from mutation, the search diversity is enlarged and the possibility of being trapped into local optimum is greatly reduced.

The rest of this paper is organized as follows: Section 2 formulates the workflow scheduling problems. Section 3 describes the original GTOA and our proposed M-GTOA algorithms. Section 4 is the experimental settings and result analysis. We finally conclude this work and indicate our

future work in Section 5.

2. PROBLEM FORMULATION

2.1. Workflow and Resource Models

A workflow can be described as a Directed Acyclic Graph (DAG) $G = \{T, E\}$, where $T = \{t_1, t_2, \ldots, t_n\}$ is a set of tasks in the workflow, and $E = \{e_{i,j} = (t_i, t_j) \mid t_i, t_j \in T\}$ is a set of directed edges representing the precedence dependencies between tasks. The edge $e_{i,j}$ indicates that task t_i is a predecessor (parent) task of t_j while t_j is a successor (child) task of t_i. t_j can be executed only if all its predecessor tasks are completed. A task without any predecessors is named as an entry task, while a task without any successors is called an exit task.

Cloud Service Providers (CSPs) usually offer users with heterogeneous resources, i.e., VMs with different types and different configurations. Each type of VM is distinguished by its processing capability in term of Million Instructions Per Second (MIPS) and price. Suppose that all the VMs are provisioned in the same cloud data center, and all the VMs take the same bandwidth, i.e., *bw*, to communicate each other. In this paper, we only consider the pay-as-you-go pricing model. Here the rental period of a VM depends on the number of minimum pricing unit time when VM is rented, instead of its real renting duration. Let $VM = \{r_1, r_2, \cdots, r_m\}$ be a set of available VMs in the cloud. Each VM $r_k = \{pc_k, uc_k, RST_k, RFT_k \mid k = 1, 2, \cdots, m\}$ is associated with four attributes, where pc_k is the processing capacity of r_k, uc_k is the unit price per time interval of r_k, and RST_k and RFT_k are the start and finish renting time of r_k, respectively.

2.2. Scheduling Formulation

Assuming a task t_i is allocated to an VM r_k. Then, the execution time $ET_{i,k}$ of t_i on r_k is calculated as:

$$ET_{i,k} = \frac{l_i}{pc_k} \quad \cdots\cdots\cdots (1)$$

where, l_i is the instruction length of t_i, and pc_k is the processing capability of r_k.

The transfer time TT_i of t_i is the sum of the time for data transmission to all of its child tasks, in the set, $succ_i$, which can be calculated as:

$$TT_i = \sum_{t_c \in succ_i} TT_{i,c} \quad \cdots\cdots (2)$$

$$TT_{i,c} = \begin{cases} \dfrac{Data_{i,c}}{bw}, & \text{if } r_k \neq r_s \\ 0, & \text{otherwise} \end{cases} \quad \cdots\cdots (3)$$

where, t_c is the child task of t_i allocated to r_s, $Data_{i,c}$ is the amount of data transferred from t_i to t_c, and *bw* is the bandwidth between r_k and r_s.

Then, the start time ST_i of task t_i can be obtained by:

$$ST_i = \max\left\{ Avail_k, \max_{t_p \in pred_i}\left\{FT_p\right\} \right\} \quad \cdots (4)$$

where, $Avail_k$ is the available time of r_k, FT_p is the finish time of t_i's parent task t_p.

The finish time FT_i of task t_i can be formulated as:

$$FT_i = ST_i + ET_{i,k} + TT_i \quad \cdots\cdots (5)$$

In this work, we define the total execution time of the workflow as *Makespan*, which can be determined by the maximum finish time among all its tasks:

$$Makespan = \max_{t_i \in T}\left\{FT_i\right\} \quad \cdots\cdots (6)$$

As mentioned above, when calculating the renting cost of an VM r_k, the renting duration of an VM depends on the number of its pricing intervals, i.e., the execution cost EC_k of r_k is calculated as:

$$EC_k = \left\lceil \frac{RFT_k - RST_k}{\Delta} \right\rceil \times uc_k \quad \cdots (7)$$

where, Δ is the minimum billing interval, $\lceil \bullet \rceil$ is a ceiling function (e.g., $\lceil 2.5 \rceil = 3$), and uc_k is the unit price per time interval of r_k.

The total execution cost of a workflow, i.e., *Cost* is computed as:

$$Cost = \sum_{k=1}^{m} EC_k \quad \cdots\cdots (8)$$

Our considered problem aims to minimize *Cost* under the deadline constraint, and it can be described as:

$$\begin{cases} \text{minimize} & Cost \\ s.t. & Makespan \leq deadline \end{cases} \quad (9)$$

3. PROPOSED ALGORITHM

3.1. Encoding and Initializing

In GTOA, one student represents a scheduling solution. As shown in Fig. 1, each dimension in the student vector denotes a task, and the value of its element stands for the VM number assigned to the corresponding task. The vector length equals to the number of tasks in a workflow.

	t_1	t_2	t_3	t_4	t_5
Student	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$X_{1,5}$
	2.1	3.7	0.4	2.0	0.9

Mapping of task to VM

$t_1 \rightarrow r_2$	$t_2 \rightarrow r_3$	$t_3 \rightarrow r_0$	$t_4 \rightarrow r_2$	$t_5 \rightarrow r_0$

Fig. 1 Encoding scheme

The population is initialized by randomly mapping tasks to its available VMs. Assume that the total number of VMs is *m*, then the process of VM allocation can be described as:

$$r_i = \lfloor random \times m \rfloor \cdot \cdot \cdot \cdot \cdot \cdot \cdot (10)$$

where, *random* is a randomly generated float point number ranging [0, 1).

3.2. Group teaching optimization algorithm

To evaluate the fitness of each student, we use the following 3 rules:

(1) If neither of two solutions satisfy the deadline constraint, then the one with smaller makespan is better.

(2) If only one solution is feasible, then the feasible one is better.

(3) If both of two solutions are feasible, then the one with lower cost is better.

After that, we sort all students according to the above rules and the best half of students are divided into the good group and the other half are in average group. During the teacher allocation phase, we select the best three students by the above comparison scheme, and calculate the teacher allocation $\mathbf{T}^t$ as:

$$\mathbf{T}^t = \begin{cases} \frac{\mathbf{x}_{\text{first}} + \mathbf{x}_{\text{second}} + \mathbf{x}_{\text{third}}}{3}, & \text{if } f(\frac{\mathbf{x}_{\text{first}} + \mathbf{x}_{\text{second}} + \mathbf{x}_{\text{third}}}{3}) \geq f(\mathbf{x}_{\text{first}}) \\ \mathbf{x}_{\text{first}}, & \text{otherwise} \end{cases} \quad (11)$$

However, if the teacher allocation is $\mathbf{x}_{\text{first}}$, the solution diversity might decrease and the search process will fall into local optima. To improve search efficiency, we change the Eq. 11 into Eqs. 12 and 13.

$$\mathbf{T}^t = \frac{\mathbf{x}_{\text{first}} + \mathbf{x}_{\text{second}} + \mathbf{x}_{\text{third}}}{3} \cdot \cdot \cdot \cdot \cdot (12)$$

$$\mathbf{T}^t = \begin{cases} \mathbf{T}^t, & \text{if } f(\mathbf{T}^t) \geq f(\mathbf{T}^{t-1}) \\ \mathbf{T}^{t-1}, & \text{otherwise} \end{cases} \cdot \cdot \cdot \cdot (13)$$

In the teacher phase, the knowledge of students within the good half can be calculated as:

$$\mathbf{x}_{\text{teacher},i}^{t+1} = \mathbf{x}_i^t + a \times \left(\mathbf{T}^t - F \times \left(b \times \mathbf{M}^t + c \times \mathbf{x}_i^t \right) \right) \cdot (14)$$

$$\mathbf{x}_{\text{teacher},i}^{t+1} = \begin{cases} \mathbf{x}_{\text{teacher},i}^{t+1}, & \text{if } f(\mathbf{x}_{\text{teacher},i}^{t+1}) \geq f(\mathbf{x}_i^t) \\ \mathbf{x}_i^t & \text{otherwise} \end{cases} \cdot \cdot (15)$$

where, F is equal to 1, a, b and c are random float point numbers ranging [0, 1). The update of average students is calculated as:

$$\mathbf{x}_{\text{teacher},i}^{t+1} = \mathbf{x}_i^t + 2 \times d \times \left(\mathbf{T}^t - \mathbf{x}_i^t \right) \cdot \cdot \cdot (16)$$

$$\mathbf{x}_{\text{teacher},i}^{t+1} = \begin{cases} \mathbf{x}_{\text{teacher},i}^{t+1}, & \text{if } f(\mathbf{x}_{\text{teacher},i}^{t+1}) \geq f(\mathbf{x}_i^t) \\ \mathbf{x}_i^t, & \text{otherwise} \end{cases} \quad (17)$$

where, d is a random float point number ranging $[0,1)$. In the student phase, the new knowledge can be obtained from self-learning and the interaction with other students, and formulated as:

$$\mathbf{x}_{\text{student},i}^{t+1} = \begin{cases} \mathbf{x}_{\text{teacher},i}^{t+1} + e \times \left(\mathbf{x}_{\text{teacher},i}^{t+1} - \mathbf{x}_{\text{teacher},j}^{t+1} \right) \\ \quad + g \times \left(\mathbf{x}_{\text{teacher},i}^{t+1} - \mathbf{x}_i^t \right), \quad \text{if } f(\mathbf{x}_{\text{teacher},i}^{t+1}) \geq f(\mathbf{x}_{\text{teacher},j}^{t+1}) \\ \mathbf{x}_{\text{teacher},i}^{t+1} + e \times \left(\mathbf{x}_{\text{teacher},j}^{t+1} - \mathbf{x}_{\text{teacher},i}^{t+1} \right) \\ \quad + g \times \left(\mathbf{x}_{\text{teacher},i}^{t+1} - \mathbf{x}_i^t \right), \quad \text{otherwise} \end{cases} \quad (18)$$

$$\mathbf{x}_{\text{student},i}^{t+1} = \begin{cases} \mathbf{x}_{\text{student},i}^{t+1}, & \text{if } f(\mathbf{x}_{\text{student},i}^{t+1}) \geq f(\mathbf{x}_{\text{teacher},i}^{t+1}) \\ \mathbf{x}_{\text{teacher},i}^{t+1}, & \text{otherwise} \end{cases} \cdot \cdot (19)$$

where, e and g are random float point numbers ranging [0, 1). It is clear from the comparison at the end of every phase that, after each update, the fitness of students will not get worse. The specific process of GTOA is shown in **Algorithm 1**.

Algorithm 1: Group Teaching Optimization Algorithm

```
Generate initial population X
Calculate current fitness values of all the individuals in X
Initialize x_best and T^0
while i < T_max do
    T^t ← Update T^{t-1} according to Eqs. 12 and 13
    Group X^t into X^t_good and X^t_ave
    for x_i^t ∈ X^t_good do
        update x^{t+1}_{teacher,i} according to Eqs. 14 and 15
    end
    for x_i^t ∈ X^t_ave do
        update x^{t+1}_{teacher,i} according to Eqs. 16 and 17
    end
    for x_i^t ∈ X^{t+1}_teacher do
        update x^{t+1}_{student,i} according to Eqs. 18 and 19
    end
    X^{t+1} ← Update X^t
    Update x_best
    i ← i+1
end
return x_best .
```

3.3. Mutation strategy

GTOA converges too quickly. As a result, it easily falls into local optima and hardly satisfies the deadline constraints in many cases. To improve GTOA's performance, we introduce a probability-based mutation strategy in the end of the student phase, and then propose a Mutation-based Group Teaching Optimization Algorithm, i.e., M-GTOA. Firstly, a random number between [0, 1), i.e., *randNum* is generated per student. If *randNum* is smaller than our predefined mutation rate, the mutation operation is used. When mutating, two random integers, i.e., *randt* and *randvm* are generated respectively, and *randt* ranging from 0 to n-1, denotes the index of task waiting for mutation, while *randvm* ranging from 0 to m-1, is the index of a new VM. The specific procedure is shown in **Algorithm 2**.

Algorithm 2: Mutation-based Group Teaching Optimization Algorithm

Generate initial population $\mathbf{X}$
Calculate current fitness values of all the individuals in $\mathbf{X}$
Initialize x_{best} and $\mathbf{T}^0$
while $i < \mathbf{T}_{max}$ **do**
  $\mathbf{T}^t \leftarrow$ Update $\mathbf{T}^{t-1}$ according to Eqs. 12 and 13
  Group $\mathbf{X}^t$ into $\mathbf{X}^t_{good}$ and $\mathbf{X}^t_{ave}$
  for $\mathbf{x}^t_i \in \mathbf{X}^t_{good}$ **do**
    Calculate $\mathbf{x}^{t+1}_{teacher,i}$ according to Eqs. 14 and 15
  end
  for $\mathbf{x}^t_i \in \mathbf{X}^t_{ave}$ **do**
    update $\mathbf{x}^{t+1}_{teacher,i}$ according to Eqs. 16 and 17
  end
  for $\mathbf{x}^t_i \in \mathbf{X}^{t+1}_{teacher}$ **do**
    update $\mathbf{x}^{t+1}_{student,i}$ according to Eqs. 18 and 19
  end
  for $\mathbf{x}^t_i \in \mathbf{X}^t_{ave}$ **do**
    if *randNum* < mutation rate **then**
      randt = *random* [0, *n*-1]
      randvm = *random* [0, *m*-1]
      $\mathbf{x}^t_i$ (*randt*) = *randvm*
    end
  end
  $\mathbf{X}^{t+1} \leftarrow$ Update $\mathbf{X}^t$
  Update x_{best}
  $i \leftarrow i+1$
end
return x_{best} .

4. EXPERIMENTAL RESULTS AND ANALYSIS

4.1. Experimental setup

All of the experiments are conducted on WorkflowSim [11], a convincing toolkit to simulate cloud workflow scheduling processes, and a Laptop with Core i7, 3.4GHz, 15.7GB RAM, Windows 11, Java 8 Standard Version V 1.8.1.

In this paper, we choose four workflows applied in different scientific areas with small and medium scales, i.e., Montage with 50 and 100 tasks, Epigenomics with 46 and 100 tasks, Inspiral with 50 and 100 tasks, and Sipht with 60 and 100 tasks. They are different in model structures, communication data, and computational characteristics, as illustrated in Fig. 2.

The parameters of VMs are all set according to Amazon EC2 [12]. In our experiments, assume the CSP offers six different types of VMs and their rental prices and processing speeds are given in Table 1. Besides, the bandwidth between VMs is 15MBps in average.

To make a comprehensive comparison, we use three-level deadline constraints, tight, moderate and relaxed and determine two baselines by HEFT. First, we only take the cheapest VM to execute all of the tasks to get the ceiling makespan M_C. Second, the makespan M_F is

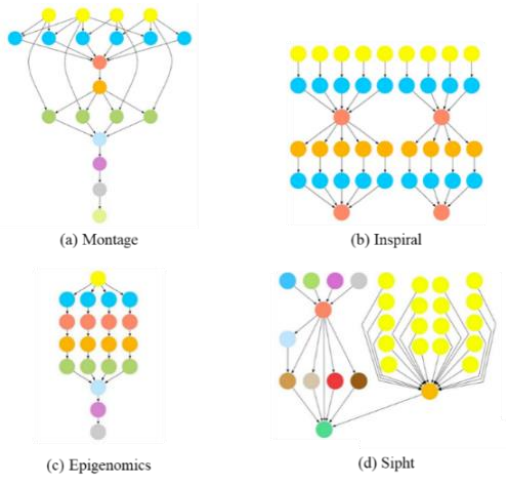


Fig. 2 The structures of realistic scientific workflows

obtained by dispatching all the tasks to the fastest VM. Third, the constraint factors λ are set as -0.02, 0.075 and 0.3 for small-scale workflows, and -0.02, 0.15 and 0.3 for the medium-scale. The deadline is calculated as:

$$deadline = M_F + \lambda \times (M_C - M_F) \quad \cdots (20)$$

Parameters of four algorithms are given in Table 2.

Table 1 Configurations of virtual machines

Index	Type	Speed (MIPS)	Price ($/h)
1	m6g.medium	4400	0.0385
2	m6g.large	8800	0.077
3	m6g.xlarge	17600	0.154
4	m6g.2xlarge	35200	0.308
5	m6g.4xlarge	70400	0.616
6	m6g.8xlarge	140800	1.232

Table 2 Parameter settings for algorithms

Algorithms	Parameters
GTOA	teaching factor F = 1, population size = 200, iteration number = 500
M-GTOA	teaching factor F = 1, mutation rate = 0.15, the population size = 200, iteration number = 500
GA	crossover rate = 1, mutation rate = 0.075, population size = 200, iteration number = 1000
PSO	$c_1 = c_2 = 2$, $\omega = 0.5$, population size = 200, the maximum iteration number = 1000
OSAM	Population size = 200, the maximum iteration number = 1000, $\beta = 3$

In this work, we compare and evaluate five algorithms from 4 aspects. To avoid randomness, we make 20 simulations for all algorithms.

The Success Rate (SR) is the first performance metric used to reflects the ability of the algorithm meeting different deadline constrains, which is defined as:

$$SR = \frac{s^*}{s} \quad \cdots (21)$$

where, s is the total number of solutions and is equal to 20 in our experiment, s^* is the number of solutions satisfying

the deadline constraints. Obviously, the higher SR, the better performance.

The average cost reflects the algorithm's performance in optimizing the cost, thus we calculate it from all feasible solutions in 20 runs.

In terms of medium-scale workflows under relaxed constraints, we also record the convergence processes of five peer algorithms. It should be noted that, to better show the convergence curve, we extend the iteration number of GTOA and M-GTOA to 1000, which is consistent with the other three algorithms.

Except for optimization performance, a high solution speed is important for real application scenarios. We also compare the running time of 5 algorithms, which reflects the efficiency to obtain the near-optimal solution.

4.2. Results and Analysis

(1) constraints satisfiability

Useful solutions should firstly meet the deadline constraints for constrained optimization problems. Fig. 3 shows the statistics about the success rate on different workflows under three constraint factors.

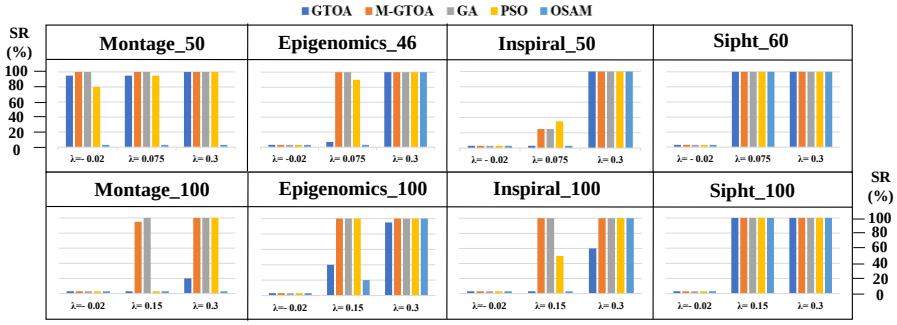


Fig. 3. Success rates of each algorithm

For tight constraints on different workflows, few algorithms can find feasible solutions except for Montage_50. In this case, M-GTOA and GA achieve a 100% SR, and OSAM gets the lowest, i.e., 0.

For moderate constraints, GA mostly obtains the highest SR, while M-GTOA achieves the second highest. Nevertheless, compared with original GTOA, M-GTOA gets a great improvement in constraint satisfiability. OSAM still performs the worst in this situation.

For relaxed constraints, all algorithms achieve SR of 100% except for GTOA and OSAM. This shows that, introducing the mutation strategy is of great significance for GTOA to escape from local optima. It should be noted that for Montages, even under relaxed constraint, OSAM cannot obtain feasible solutions.

(2) Cost optimization

To assess the performance of cost optimization of five algorithms, Figs. 4-7 illustrate the results from the average cost of feasible solutions in 20 runs on each workflow under three constraints.

For both scales of Montage workflows, M-GTOA performs the best under tight and moderate constraints and the second best under relaxed constraints. Although GTOA achieves better results sometimes, it might not meet constraints as shown in Fig. 3. Noting that the blue block, e.g., Montage_100, $\lambda = 0.15$ in Fig. 4, means the approach has never met the constraint, thus there is no available data to calculate the average cost.

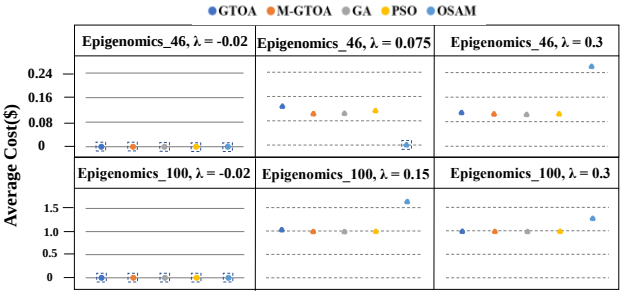


Fig. 5 The average costs on Epigenomics

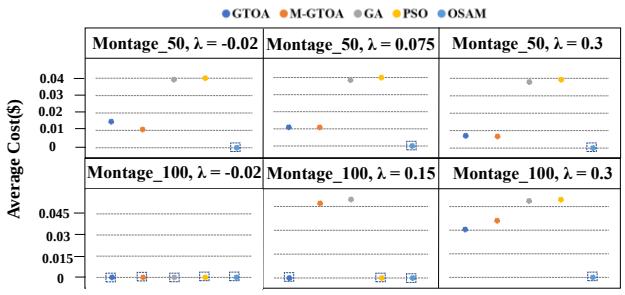


Fig. 4 The average costs on Montage

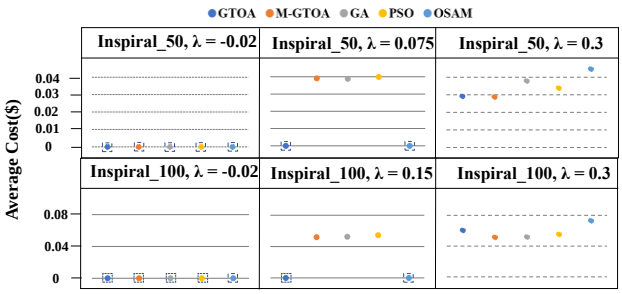


Fig. 6 The average costs on Inspiral

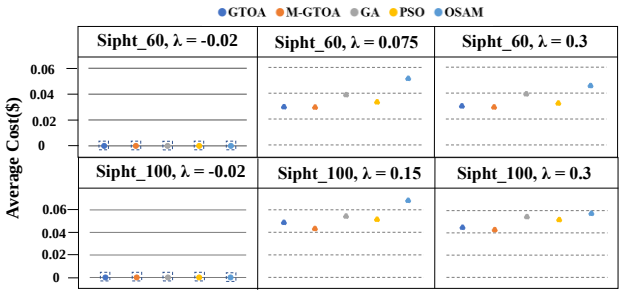


Fig. 7 The average costs of algorithms over Sipht

For Epigenomics, M-GTOA and GA show great performances, and OSAM is the worst for allcases. Specially, four approaches, except OSAM, perform similarly under the relaxed constraint on the medium scale.

For small scale Inspiral workflows, the optimization performances are similar under tight constraints as all algorithms cannot find feasible solutions and thus focus on optimizing makespan. In terms of moderate constraints, there is no available data from GTOA and OSAM, and the performances of the other three are also similar. GTOA and M-GTOA perform better than the other two under the relaxed constraint and meanwhile, M-GTOA is the best. As for medium scale Inspiral, in most cases, GA and M-GTOA perform the best, whereas OSAM the worst.

For Sipht workflows, in most cases, M-GTOA shows the best performance while GTOA the second best.

(3) Convergence

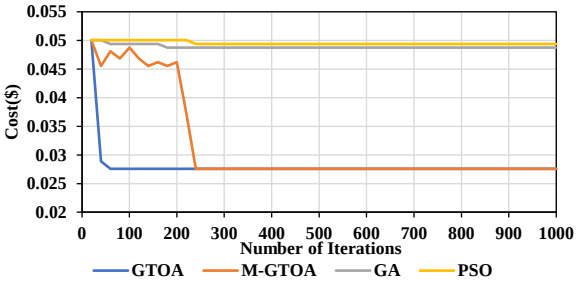


Fig. 8 The iteration processes on medium-scale Montage

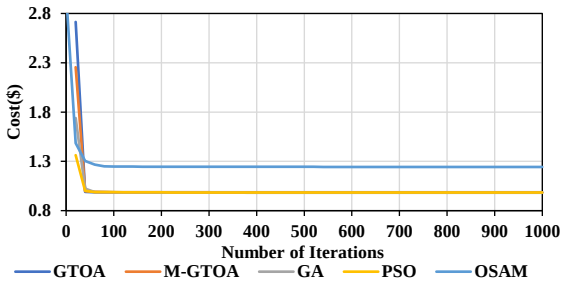


Fig. 9 The iteration processes on medium-scale Epigenomics

The four iterative processes for each workflow are demonstrated in Figs. 8 to 11. Noting that, we record the resulting cost every 20 iterations under the maximum iteration number 1000. Nevertheless, according to Fig. 3, for medium-scale Montage, OSAM fails to find feasible solutions under relaxed constraints and therefore the iteration process of OSAM is not given in Fig. 10.

It can be observed from Figs. 8 to 11 that, M-GTOA converges more slowly than GTOA. But due to the characteristics of GTOA, both of them usually find the optimal solutions within 200 iterations. OSAM also converges at a relatively fast speed compared with GA and PSO. Specifically, for GA and PSO, the optimal solutions are usually found after 200 iterations, even 300. It can be seen that, GTOA and OSAM converge fast so that their optimization performances are not as satisfying as that of the other three.

(4) Running time

The algorithm efficiency is reflected by its running time. To obtain their running time, we run 20 times on each workflow under three constraints and get the average running time, as shown in Tables 3 and 4.

According to Table 3, for small-scale workflows, besides the case of Sipht, OSAM obtains the shortest running time in all cases. GTOA gets the second, while GA is the slowest in many cases. In contrast, OSAM obtains the longest running time in Sipht. Moreover, it is predictable that the running time of M-GTOA is larger than that of GTOA since we introduce a mutation strategy bringing more update operations to M-GTOA.

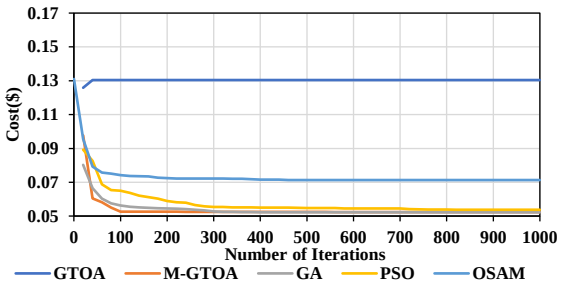


Fig. 10 The iteration processes on medium-scale Inspiral

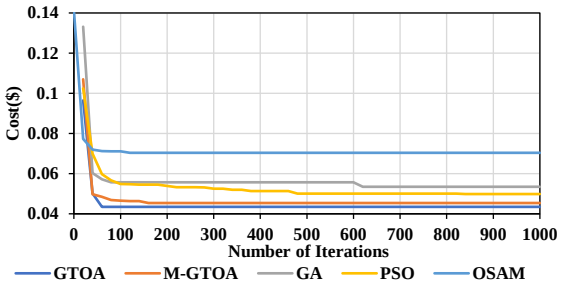


Fig. 11 The iteration processes on medium-scale Sipht

In terms of medium-scale workflows, the situation is quite similar as the small-scale's. OSAM is the fastest in Montage, Epigenomics and Inspiral, while GTOA is the fastest in Sipht. Nevertheless, M-GTOA always shows a better performance compared with GA and PSO.

Table 3 Running time of small-scale workflow

Workflow	Constraint Factor	Algorithm				
		GTOA	M-GTOA	GA	PSO	OSAM
Montage_50	λ = - 0.02	2.84405	3.13525	3.394	3.3989	**2.36565**
	λ = 0.075	2.79075	2.9922	3.3085	3.09565	**2.11795**
	λ = 0.3	2.86685	3.01375	3.54805	3.45335	**2.32865**
Epigenomics_46	λ = - 0.02	2.78025	2.5825	3.234	2.7373	**1.73219**
	λ = 0.075	2.70525	2.53325	2.8021	2.7822	**1.7039**
	λ = 0.3	2.58525	2.6269	2.97015	2.91585	**1.8685**
Inspiral_50	λ = - 0.02	2.7086	2.718	3.08985	2.9775	**1.92055**
	λ = 0.075	2.825	2.9524	3.24915	2.83625	**1.9442**
	λ = 0.3	2.86865	3.03785	3.056	2.6871	**1.9827**
Sipht_60	λ = - 0.02	**3.4067**	3.76045	4.2464	3.7365	4.52805
	λ = 0.075	**3.41555**	3.5417	4.0496	3.57535	4.571
	λ = 0.3	**2.84405**	3.13525	3.394	3.3989	4.57625

Table 4 Running time of medium-scale workflow

Workflow	Constraint Factor	Algorithm				
		GTOA	M-GTOA	GA	PSO	OSAM
Montage_100	λ = - 0.02	7.6231	8.43065	7.86715	8.33495	**6.7167**
	λ = 0.15	7.05345	8.31625	7.4072	7.84335	**6.20695**
	λ = 0.3	7.4409	8.52735	7.8418	7.7449	**6.2494**
Epigenomics_100	λ = - 0.02	4.5322	4.88685	4.6508	5.26285	**3.6853**
	λ = 0.15	4.31875	4.7072	4.567	4.9565	**3.3739**
	λ = 0.3	4.2752	4.76805	4.98145	4.96115	**3.55065**
Inspiral_100	λ = - 0.02	4.51345	4.8649	5.3667	5.6944	**3.6622**
	λ = 0.15	4.70945	4.8046	5.3005	5.43445	**3.6839**
	λ = 0.3	4.65485	4.93625	4.84825	5.21685	**3.5409**
Sipht_100	λ = - 0.02	**5.4112**	5.44915	6.88045	6.88885	7.62195
	λ = 0.15	**5.250048**	5.46585	5.8519	6.3358	7.6915
	λ = 0.3	**5.189**	5.2527	6.01955	6.3681	7.7294

5. Conclusion

In this paper, we adopt GTOA proposed for continuous optimization problems into discrete optimization, like workflow scheduling problems with small and medium scales. GTOA needs only two essential common parameters, the population size and the termination criterion, which makes it more conveniently used. To better balance between exploration and exploitation, and escape from local optima, we put forward a M-GTOA algorithm to minimize the workflow execution cost under deadline constraints by embedding a mutation operation into GTOA's update process to improve its search diversity. We then compare our proposed GTOA-based optimizer, M-GTOA with GA, PSO, OSAM and the original GTOA to evaluate its effectiveness in 4 performance metrics. The experimental results show the superiority of M-GTOA over all its peers.

In future work, we plan to conduct further research on M-GTOA, i.e., optimizing multiple objectives, such as the makespan, cost and energy consumption at the same time. Moreover, as seen in Section 4, in terms of the performance of cost optimization, M-GTOA does not always perform the best, and it will be promising to combine M-GTOA with other meta-heuristics.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61836001; and in part by the National Key Research and Development Program of China under Grant No. 2018YFB1003700.

REFERENCES

[1] Y. Xia, M. Zhou, X. Luo, S. Pang and Q. Zhu, "Stochastic Modeling and Performance Analysis of Migration-Enabled and Error-Prone Clouds," IEEE Transactions on Industrial Informatics, vol. 11, no. 2, pp. 495-504, 2015, doi: 10.1109/TII.2015.2405792.

[2] H. Topcuoglu, S. Hariri and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, 2022, doi: 10.1109/71.993206.

[3] Y. Yuan, X. Li, Q. Wang, X. Zhu, "Deadline Division-based Heuristic for Cost Optimization in Workflow Scheduling," Information Sciences, vol. 179, no. 15, pp. 2562-2575, 2009, doi:10.1016/j.ins.2009. 01.035.

[4] J. Meena, M. Kumar and M. Vardhan, "Cost Effective Genetic Algorithm for Workflow Scheduling in Cloud Under Deadline Constraint," IEEE Access, 2016, vol. 4, pp. 5065-5082, doi: 10.1109/ ACCESS. 2016.2593903.

[5] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioningand Scheduling Algorithm for Scientific Workflows on Clouds," IEEE Transactions on Cloud Computing, vol. 2, no. 2, pp. 222-235, 2014, doi: 10.1109/TCC.2014.2314655.

[6] V. Vinothina, "Scheduling Scientific Workflow Tasks in Cloud Using Swarm Intelligence," 2017 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), 2017, pp. 1-5, doi: 10.1109/ ICCTAC.2017. 8249991.

[7] H. Li, D. Wang, G. Xu, Y. Yuan and Y. Xia, "Improved Swarm Search Algorithm for Scheduling Budget-Constrained Workflows in the Cloud," Soft Computing, vol. 26, pp. 3809-3824, 2022, doi: 10.1007/s00500-022-06782-w

[8] R. V. Rao, V. J. Savani, D. P. Vakharia, "Teaching–Learning-Based Optimization: a Novel Method for Constrained Mechanical Design Optimization Problems," Computer-Aided Design, vol. 43, no. 3, pp. 303–315, 2011, doi: 10.1016/j.cad. 2010.12. 015.

[9] A. Sadollah, H. Sayyaadi, A. Yadav, "A Dynamic Metaheuristic Optimization Model Inspired by Biological Nervous Systems: Neural Network Algorithm," Applied Soft Computing, vol. 71, pp. 747–782, 2018, doi: 10.1016/j.asoc.2018.07.039.

[10] Y. Zhang, Z. Jin. "Group Teaching Optimization Algorithm: A Novel Metaheuristic Method for Solving Global Optimization Problems," Expert Systems with Applications, vol. 148, pp. 113246, 2020, doi: 10.1016/j.eswa.2020.113246.

[11] W. Chen and E. Deelman, "WorkflowSim: A toolkit for Simulating Scientific Workflows in Distributed Environments," 2012 IEEE 8th International Conference on E-Science, 2012, pp. 1-8, doi: 10.1109/eScience.2012. 6404430.

[12] Amazon. "Amazon ec2 pricing". https:// amazonaws-china.com/cn/ec2/pricing/ on-demand/.