Multi-objective workflow scheduling based on genetic algorithm in cloud environment

Xuewen Xia*[c,d], Huixian Qiu*[a,b], Xing Xu[c,d], Yinglong Zhang[c,d]

[a] *College of Computer, Minnan Normal University, Zhangzhou, 363000, Fujian, China*
[b] *Key Laboratory of Data Science and Intelligence Application, Minnan Normal University, Zhangzhou, 363000, Fujian, China*
[c] *College of Physics and Information Engineering, Minnan Normal University, Zhangzhou, 363000, Fujian, China*
[d] *Key Lab of Intelligent Optimization and Information Processing, Minnan Normal University, Zhangzhou, 363000, Fujian, China*

Abstract

In recent years, cloud computing plays a crucial role in many real applications. Thus, how to solve workflow scheduling problems, i.e., allocating and scheduling different resources, under the cloud computing environment becomes more important. Although some evolutionary algorithms (EAs) can solve workflow scheduling problems with a small scale, they show some disadvantages on larger scale workflow applications. In this paper, a multi-objective genetic algorithm (MOGA) is applied to optimize workflow scheduling problems. To enhance the search efficiency, this study proposes an initialization scheduling sequence scheme, in which each task's data size is considered when initializing its virtual machine (VM) instance. Relying on the initial scheduling sequence, a proper trade-off between the makespan and the energy consumption, which are two optimization objectives in this study, can be achieved. In the early evolution stage, traditional crossover and mutate operators are performed to keep the population's exploration. On the contrary, the longest common subsequence (LCS) of multiple elite individuals, which can be regarded as a favorable gene block, is saved during the later evolution stage. Based on the LCS, the probability of some favorable gene blocks being destroyed will be reduced when performing the crossover operator and the mutate operator. Hence, the integration of the LCS in GA can satisfy different requirements in different evolution stages, and then to attain a balance between the exploration and the exploitation. Extensive experimental results verify that the proposed GA combined with LCS, named as GALCS in this paper, can find a better Pareto front than the ordinary GA as well as other state-of-the-art algorithms. Furthermore, effectivenesses of the new proposed strategies are also verified by a set of experiments.

Keywords: Task scheduling, Genetic algorithm, Scheduling sequences, Multi-objective optimization.

*Corresponding authors: Xuewen Xia, Huixian Qiu, Email addresses: xwxia@whu.edu.cn, 1512150486@qq.com

1. Introduction

In recent years, cloud computing is widely popular in e-commerce, image processing, astronomy physics, and social networks [2, 19, 34]. With the rapid increase number of clients, cloud service providers should design an efficient and secure system for resource allocation and scheduling [28]. Generally, a system is considered efficient if it can provide the best performance for all applications running within it under resource constraints [17]. Therefore, to meet the requirements, it is necessary to manage resources effectively and schedule tasks reasonably. Generally, the cloud computing service types can be divided into three categories: infrastructure as a service (IaaS) [44], platform as a service (PaaS) [43], and software as a service (SaaS) [41]. IaaS refers to a service model which provides cloud computing infrastructure as a service through the network and charges according to the actual use or occupation of resources by users. In IaaS, users choose an appropriate virtual machine (VM) according to their own needs. PaaS is a cloud service mode which provides users with an environment to satisfy users' software development, debugging and operation as a service. Users can satisfy their own needs without paying attentions to the operation of underlying infrastructure and systems. SaaS is a new way for users to obtain software services. It does not require users to install software products on their own computers or servers, but directly obtain services with corresponding software functions from special providers through the network. To improve users' experience, how to satisfy their distinct requirements to the greatest extent has become a primary task for cloud service providers. Compared with PaaS cloud and SaaS cloud, IaaS cloud can be regarded as the most suitable model for performing workflow.

Workflow is a common computational model of scientific applications [40]. The main objective of the model is designing a set of logics and rules of how to organize tasks in workflow. It is widely accepted that a large-scale workflow should be deployed in a high-performance distributed computing environment and then to be executed quickly. Obviously, the cloud computing can provide elastic resources to solve the problem since the high-performance computing resources can be obtained and released as needed.

Workflow scheduling is a process of assigning available resources to committed applications for given periods, and then enabling all the applications to efficiently utilize the resources aiming to maximize the quality of service (QoS) [26]. A scheduling algorithm usually assigns each task to a VM in the order of a scheduling sequence. Therefore, the task scheduling sequence has a significant impact on the execution time and cost of a workflow application. Generally, a scheduling problem is a NP-hard optimization problem, which means that the problem cannot be solved in polynomial time. Thus, workflow scheduling is a challenging task in the cloud environment. Although some deterministic algorithms can attain promising results in a small scale scheduling problem, they offer unfavorable performance on a large scale problem. At present, evolutionary algorithms (EAs) and heuristic algorithms, which have been successfully applied in many applications, are widely considered as two types of promising methods for the task scheduling problems.

EAs [20] are a type of search algorithms based on biological evolution mechanisms such as natural selection and natural genetics. The evolutionary selection strategy of an EA has

a great influence on the convergence of the algorithm. When the problem size increases, the convergence of the algorithm can be improved by using a better evolutionary selection strategy. A heuristic algorithm is a rule-based algorithm, which can quickly obtain solutions for the problem. However, the solution space searched by this method is limited. Thus, when the scale of the problem increases, its success in solving the problem decreases due to that the effect of this method is closely related to heuristic rules.

In this paper, we propose a genetic algorithm (GA) [16] based method to solve the multi-objective workflow scheduling problem. This approach combined GA with the longest common subsquence (LCS) algorithm, termed GALCS, aims to record the favorable gene blocks and to improve the performance of GA. Moreover, to enhance the search efficiency, each task's data size is considered when initializing its VM instance rather than randomly assigning a VM to the task.

Contributions of this paper are summarized as follows.

(1) An initialization method for scheduling sequence based on task data sizes is proposed. In the initialization method, a task with greater data volume is scheduled on those VMs who have a higher processing capacity rather than allocated in randomly selected VMs. Based on the strategy, the initial population performs better than the random initialization method, in terms of makespan.

(2) The LCS of multiple elite individuals, which can be regarded as a favorable gene block, is saved during the evolution stage process.

(3) In the early evolution stage, traditional crossover and mutate operators are applied to enhance the population's exploration. On the contrary, to improve the convergence speed in the later evolution stage, some promising gene blocks of an individual can be saved with a higher probability when performing the crossover and mutate operators.

The remainder of this paper is organized as follows. Section 2 introduces the model and the basic elements of the scheduling problem and outlines current cloud workflow scheduling algorithms. Section 3 presents the details of the proposed approach. Experimental results are described and discussed in Section 4. Finally, we conclude our work and give insight into future works in Section 5.

2. Related work

This section firstly introduces the workflow model and the VM model in the cloud environment, and then describes the multi-objective workflow scheduling problem in this study.

2.1. Workflow scheduling problem

2.1.1. Workflow model

The topology of a workflow is usually a directed acyclic graph (DAG) [48], which can be expressed as a tuple: $G = < T, E >$, where $T = \{t_i \mid i \in n\}$ represents a set of tasks, n represents the number of tasks; $E = \{e_{ij} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n\}$ is a set of constraints relation between tasks, the value of the edge e_{ij} is c_{ij}, which means communication requirements of task t_i and task t_j.

In DAG, a task without any precursors task is the entry task t_{entry}. On the contrary, a task without any successors task is called exit task t_{exit}. Note that, there may be one or more t_{entry} and t_{exit} in a DAG. L_i represents the service length of task t_i. In this study, succ(t_i) and pred(t_i) are two task sets denoting the direct successor tasks and precursor tasks of the task t_i, respectively. The definition of the two task sets are defined as Eq. (1) and Eq. (2), respectively.

$$succ(t_i) = \{t_j \mid e_{ij} \in E, t_i, t_j \in V\} \tag{1}$$

$$pred(t_i) = \{t_j \mid e_{ji} \in E, t_i, t_j \in V\} \tag{2}$$

When a task has more than one precursors, the task cannot be executed until all its precursors tasks have been executed and all communication requirements of the task have been received.

Fig. 1 shows an example of workflow with 5 tasks, where t_1 is the entry task and t_5 is the exit task. The successors of t_1, i.e., succ(t_1), are t_2, t_3, and t_4, while the precursors of t_5, i.e., pred(t_5), are t_2, t_3, and t_4.

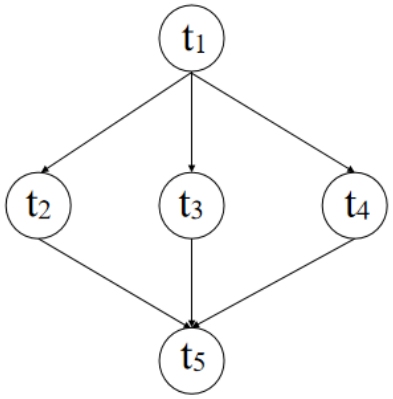


Fig. 1: An example workflow with 5 tasks.

2.1.2. Cloud resource model

A cloud model consists of a data center that offers different kind of VM types. A heterogeneous cloud computing system with m VM types can be defined as:$< P, W, Q, B >$, where $P = \{P_i \mid 1 \leqslant i \leqslant m\}$ is a set of heterogeneous VM instances, and $W = \{W_i \mid 1 \leqslant i \leqslant m\}$ represents a set of the execute capacity of VMs. If task t_i is executed in the instance P_j, the execute time of t_i is:

$$exe(t_i, P_j) = \frac{L_i}{W_j} \tag{3}$$

where L_i represents the workload of task t_i. The communication time between t_i and t_j is defined as Eq. (4), where B is the average network bandwidth between different instances.

If two tasks are executed on the same instance, their communication time is 0.

$$ct(i,j) = \frac{c_{ij}}{B} \tag{4}$$

2.1.3. Basic conception

This section introduces some basic concepts of workflow. Task makespan is a major indicator of scheduling performance. In a DAG task graph, the constraint relationship among different tasks must be considered when calculating task execution time. The earliest start time of task t_i on the executed instance P_j, denoted as $EST(t_i, P_j)$, is defined as Eq. (5):

$$\begin{cases} EST(t_{entry}, P_j) = 0 \\ EST(t_i, P_j) = \max\limits_{t_a \in pre(t_i)} \{EFT(t_a, p_b) + ct(a,i)\} \end{cases} \tag{5}$$

where $EFT(t_a, P_b)$ is the earliest finish time of task t_a executed in the instance P_b; and task t_a is one of the precursors task of the task t_i. If task t_i is t_{entry}, then the $EST(t_i, P_b)$ is 0. Otherwise a task cannot be executed before any its precursors tasks, so the $EST(t_i, P_j)$ is the max EFT of its precursors tasks adds up with communication time between them. The earliest finish time of task t_i executed in instance P_j, denoted as $EFT(t_i, P_j)$, is the sum of $EST(t_i, P_j)$ and the executed time of t_i, recorded as $exe(t_i, P_j)$. Thus, the $EFT(t_i, P_j)$ is computed as Eq. (6):

$$EFT(t_i, P_j) = EST(t_i, P_j) + exe(t_i, P_j) \tag{6}$$

According to the above discussions, the makespan of a scheduling sequence can be defined as Eq. (7):

$$makespan = EFT(t_{exit}, P_j) \tag{7}$$

Commonly, the energy consumption produced by instances takes a big part of the energy consumption produced by a computer system. In this study, the VM model is power-aware, which uses the power consumption under the current CPU utilization of the physical machine as the real-time power consumption. For each instance P_i, there are two states, i.e., running state and idle state. Correspondingly, we use Q_i and $Q_{i,idle}$ to describe the average running power and the idle power of P_i, respectively. In a workflow scheduling problem, the energy consumption consists of runtime consumption and idle time consumption. The runtime energy consumption of instance P_j, denoted as RC_j, is computed as Eq. (8):

$$RC_j = \sum_{i=1}^{n} U_{ij} * exe(t_i, P_j) * Q_j \tag{8}$$

where U_{ij} represents whether task t_i is executed on instance P_j or not. Concretely, U_{ij} has only two values, 1 or 0, which mean that the task t_i is or is not executed on instance P_j, respectively. $exe(t_i, P_j)$ is the execute time of task t_i executed on instance P_j. Q_i is the instance's runtime energy. Thus all the processing energy consumption of instances, ER is:

$$ER = \sum_{j=1}^{m} RC_j \tag{9}$$

Furthermore, when an instance runs at low power in idle time, it also produces energy consumption. Generally, the idle time energy consumption is the instance's idle time multiplied by the instance's idle power. Thus, the idle time energy consumption of an instance P_j can be defined as:

$$IC_j = (makespan - \sum_{j=1}^{n} U_{ij} * exe(t_i, P_j)) * Q_{j,idle} \tag{10}$$

where $makespan$ is the total finish time of the scheduling sequence, and $Q_{i,idle}$ is the idle power of the instance P_j. Thus, all the idle time energy consumption of EI is:

$$EI = \sum_{j=1}^{m} IC_j \tag{11}$$

Based on the above mentioned discussion we known that a system's total energy consumption mainly contains of the runtime energy consumption and idle time energy consumption. Hence, the total energy consumption can be defined as Eq. (12).

$$energy = ER + EI \tag{12}$$

2.1.4. Cloud scheduling problem description

Base on the above analyses, this study focuses on finding a compromise solution which can minimize the makespan and energy consumption. A solution of the scheduling problem is expressed as a scheduling sequence that satisfies dependencies on all the tasks. Typically, a multi-objective workflow scheduling problem in this paper is defined as Eq. (13):

$$\begin{cases} min\{makespan, energy\} \\ s.t. \sum_{j=1}^{m} U_{ij} = 1, i = 1, 2, ..., n \\ EFT(t_i) \leqslant EST(t_j), t_i \in pred(t_j) \end{cases} \tag{13}$$

The constraint $\sum_{j=1}^{m} U_{ij}, i = 1, 2, ..., n$ ensures all tasks are scheduling and each task to be scheduled only once, and the constraint $EFT(t_i) \leqslant EST(t_j), t_i \in pred(t_j)$makes sure that a scheduling solution in accordance with the constraint relation between task t_i and task t_j.

2.2. Background

The workflow scheduling problem is a typical NP-complete problem [33]. Thus, it's difficult for a deterministic algorithm to find an optimal solution while the scale of the problem is enormous. Hence, current research typically adopts simple heuristic algorithms [12, 36, 23] and evolutionary algorithms [25, 24, 31] to solve the problems. The former is a kind of static scheduling. These strategies can get a solution more quickly using given scheduling rules. The latter uses some evolutionary operators to find a better solution, such as GA, particle swarm optimization (PSO), and ant colony optimization (ACO). Compared

with heuristic algorithms, evolutionary algorithms can get a high-quality solution because they can explore more solutions through iterative evolution.

In the current research, many QoS metrics, such as makespan, energy consumption, cost and security, are regarded as optimization objectives of a scheduling problems [35]. In this section, we will briefly review some study of the problem. According to the type of algorithms applied in scheduling problems, the study can be divided into two categories: simple heuristics algorithms-based scheduling approaches and evolutionary algorithms-based scheduling approaches.

2.2.1. Simple heuristic algorithms based scheduling approaches

Generally, a typic scheduling algorithm has two phases. The one phase is to determine priorities of all tasks and rearrange a scheduling sequence. The other one phase is to traverse the scheduling sequence and then to select an appropriate processor to execute the tasks. For instance, Heterogeneous Earliest Finish Time (HEFT), as a well known listed scheduling algorithm, can get the solution with a low time complexity [30]. In HEFT algorithm, different tasks are given different priority values through an upward rank. Based on the priority values, the tasks are assigned to different processors intending to minimize the tasks' EFT. Critical Path on Processor (CPOP) [30] sharing similar phases as HEFT considers an upward rank and a downward rank together to calculate the priority values of the tasks. In the processor selecting phase, CPOP allocates a critical task to a critical processor and assigns other non-critical tasks to other processors aiming to finish all the tasks in the shortest time.

From the above discussions, it can be observed that a task's priority plays an important role in the list algorithms. HEFT and CPOP can get a solution after a priority-based traversal, and the completion time of them is fast. However, both HEFT and CPOP select a task only based on the tasks' minimum EFT without considering its successors. If there are heavy communication requirements between the task and its successor, the finish time of them will be extended when the task and its successor are assigned to different processors. To overcome the shortcoming, a kind of algorithms based on a predicted strategy was proposed. For instance, Predict Earliest Finish Time (PEFT) [1] uses an optimistic cost table (OCT) to define a rank of each task. PEFT needs to consider the OCT value of its successor task when calculating the OCT value of the current task. Although PEFT has the same complexity as HEFT, it does not consider tasks' executed time in the optimistic processor. Thus, if a task with a large service length is assigned to a low-capacity processor, the total finish time will be influenced.

However, all of the above mentioned algorithms only consider the makespan as an optimization objective. Hence, to overcome the weak practicality of the algorithms, a Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT) [9], as an extension of the HEFT workflow scheduling algorithm, considers both makespan and cost as two optimization objectives. In MOHEFT, all the tasks are mapped to all the processors aiming to extend the set of the target solution. This strategy can expand the number of candidate solutions that more than we want. And it uses crowding distance selection to choose the optimal solution. Like HEFT, in the process of scheduling, the arrangement of tasks in MOHEFT is never changed. This may lead to MOHEFT will not get the optimal Pareto front in the final when

choosing the optimal solution. Ritu Garg et al. [10] proposed reliability and energy-efficient workflow scheduling (REEWS) to minimize energy consumption and maximize the reliability of the workflow execution while sustaining the user-specified QoS constraint. Haidri et al. [13] proposed a workflow strategy in order to minimize the makespan and cost while meeting the deadline. The Cost Effective Deadline Aware (CEDA) algorithm selects the task which has the highest upward rank and dispatches a selected task to the cheapest instance considering the acquisition delay. Fully decentralized proximity-aware workflow-scheduling policy (FMProx) [11] introduces a method which considers the load-balancing policy, QoS constraints of each workflow and proximity of selected peers to schedule scientific workflows in a fully decentralized manner and with low overhead.

The above introduced heuristic algorithms are all based on the scheduling rules, and most scheduling rules only consider the makespan. How to determine the best scheduling sequence is not clear in multi-objective optimization problems.

2.2.2. Evolutionary algorithms based scheduling approaches

In recent years, various EAs have been successfully applied in scheduling problems. For instance, Amandeep Verma and Sakshi Kaushal [35] proposed a non-dominance sort-based Hybrid Particle Swarm Optimization (HPSO) algorithm to address the workflow scheduling problem on the cloud. In HPSO, an elite archive is adopted to store non-dominated particles found during the search process. In each generation, a density perimeter is used to compare a particle's current position with historical best position of the population, and then to update achieved optimal particle. Zhu et al. [47] adopted an evolutionary multi-objective optimization-based algorithm to minimize both makespan and cost simultaneously, in which two novel crossover and mutation operations are introduced to effectively explore the whole search space. Zhou et al. [46] developed a delay transmission mechanism to optimize the execution time of workflow and energy consumption. The algorithm is derived from GA and implements a new encoding scheme. Rodriguez and Buyya [27] introduced a resource provisioning and scheduling strategy for scientific workflows. They used a PSO to minimize the workflow execution cost while meeting deadline constraints.

In those EAs mentioned above, the initialization of the population is random. In other words, priorities of tasks and load of processors are not considered. For example, if a task with a heavy service length and a high priority is initially assigned to a processor with low processing capacity, it may cause the makespan to be longer. Thus, how to select an appropriate processor for each task in the initialization phase also needs to be considered.

Li et al. [22] adopted a Multi-Objective Memetic Algorithm (MOMA) to address the initialization problem in multi-objective scheduling optimizations. MOMA propoes a new processor initialization allocation method based on tasks' priority. To improve the exploitation ability, MOMA uses a memetic local search metric according to related information on local structure. However, the crossover and mutate in MOMA are only operated in a same layer of DAG. This may reduce the scope of crossover and variation. Wang and Zuo [39] combined PSO with idle time slot-aware rules to minimize the makespan of the workflow scheduling problem. They used a new encoding strategy to map different processors to tasks. Thus appropriate processors can be assigned to different tasks. However, the improvement

of optimization ability is not considered in the study. In 2017, an adaptive ACO-based cloud workflow scheduling algorithm was proposed by Wu et al. [42]. They firstly distribute the constraint to each task of the workflow. Then they used the proposed probabilistic upward rank to order tasks. Recently, some studies [6, 3, 37] used GA combined with other strateges to improve the performance of workflow scheduling. Extensive experiments show that the GA-based algorithm is beneficial for multi-objective workflow scheduling. Zhuo et al. [45] designed some novel schemes of evolutionary techniques to focus on three objectives: cost, makespan and system reliability. To minimize the energy consumption while meeting the deadline constraint, Indrajeet Gupta et al. [29] presented an energy efficient workflow scheduling (EEWS) algorithm inspired from hybrid chemical reaction optimization algorithm.

In this paper, we take both task scheduling location and execution sequence into consideration in the encoding scheme. An improved GA combined with LCS is proposed to enhance the search efficiency of the algorithm so as to accelerate convergence and get a better Pareto front.

3. Proposed approach

Generally speaking, a multi-objective optimization problem (MOP) has multiple objectives which need to be optimized simultaneously. A typical MOP can be defined as Eq. (14):

$$min \quad F(x) = (f_1(x), f_2(x)..., f_k(x))^T \tag{14}$$

where $x \in X$ is a candidate solution in the decision space X, and $f_1(x), f_2(x), ..., f_k(x)$ are k conflicting objective functions.

In a MOP, there is no single optimal solution for all objectives. In this kind of problem, the desired solution is considered to be a set of potential solutions optimal for one or more objectives. This set is called Pareto optimal set. Some Pareto concepts used in MOP are as follows. For $x_1, x_2 \in X$, x_1 is said to dominate x_2 if and only if

$$\forall i : f_i(x_1) \leqslant f_i(x_2) \wedge \exists j : f_j(x_1) < f_j(x_2) \tag{15}$$

The decision vector x_1 dominates x_2, which means that all objectives of x_1 are no worse than x_2, and x_1 is strictly superior to x_2 in at least one objective. When the decision vector x_1 is not dominated by any other decision vector in the set, it is called Pareto optimal. The set of all Pareto optimal solutions in the objective space is called the Pareto front. In this study, the workflow scheduling problem is a typical MOP, in which the energy consumption and the makespan are two objectives that need to be optimized [21].

The steps of multi-objective GA (MOGA) mainly include encoding, population initialization, fitness value evaluation, selection, crossover and mutate [14]. For a multi-objective workflow scheduling problem studied in this paper, the fitness of each individual is measured by makespan and energy consumption, which are defined as Eq. (7) and Eq. (12), respectively. The specific method of other steps in MOGA designed in our study will be introduced in detail below.

3.1. Encoding scheme and initialization

When applying GA to solve a specific problem, the first issue needs to be dealt with is to determine an effective chromosome encoding according the characteristics of the problem [32]. In this study, scheduling of tasks includes determining the scheduling order of tasks and mapping between tasks to target processors. Thus, a chromosome should contain two parts: one part determines the scheduling order of the tasks and the other one part represents the mapping of tasks to the VM instances. For a workflow with n tasks, we use a set $X = \{x_1, x_2, ..., x_n\}$ to represent the target instance of the tasks. The value of x_i represents the index of the target instance. The value range of each element in X is 1 to m, m is the total number of instances. We use a set $Y = \{y_1, y_2, ..., y_n\}$ to represent the execution sequence of tasks. y_i is one of the task of the workflow. The set Y must conform to the precedence constraints of the workflow. Therefore, the task t_{entry} must be ranked first and the task t_{exit} must be ranked last in the set Y. Fig. 2 shows an example of an encoding scheme for the multi-objective workflow scheduling in Fig. 1.

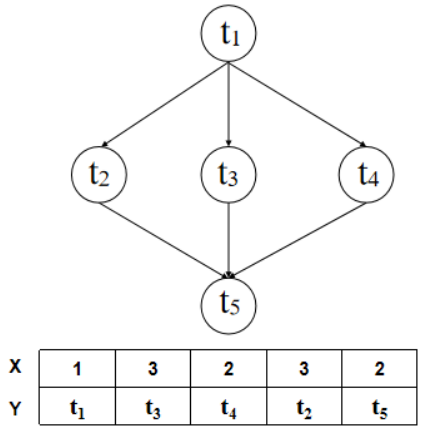


Fig. 2: A chromosome encoding for the workflow in Fig. 1.

As shown in Fig. 2, the number of instances is assumed to be 3. In this example, the value of Y is $[t_1, t_3, t_4, t_2, t_5]$, which representing the scheduling sequence of tasks is: $t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow t_2 \rightarrow t_5$. Fig. 2 also gives the mappings from the tasks to the processors. For example, the value of X, Y in dimention 1 are 1 and t_1, respectively, which means task t_1 will be scheduled to the instance P_1.

For the task scheduling order initialization, since tasks must meet certain precedence constraints, we use a set S to record the current sortable tasks, and then a task is randomly selected from the set for sorting. A sortable task is a task without predecessors or all of its predecessors have been sorted. We can continue to select the next task according to this method until a feasible task sequence is formed.

For the execution instance of the tasks, most of the previous algorithms use a random initialization method because it's not easy to determine the most appropriate VM instance for tasks. If many data-heavy tasks are assigned to instances with low processing capacity,

Algorithm 1 Chromosome Initialization

Require: The DAG of workflow, the VM instance pool;
Ensure: The scheduling solution $R = (X, Y)$.

initialize the length of chromosome n;
let S and Y be new sets;
Y=$\{t_{entry}\}$, $T = T - \{t_{entry}\}$, S=$\emptyset$;
while T$\neq \emptyset$ **do**
    for $t_i \in T$ **do**
        if $pred(t_i)$ in Y **then**
            $S = S \cup t_i$;
        end if
    end for
    randomly select a task t_k from S;
    $T = T - \{t_k\}$, $Y = Y \cup \{t_k\}$, S=$\emptyset$;
end while
calculate the average of tasks' volume $avgL$;
let p be the index of the max value of W;
$index = 1$;
for t_i in Y **do**
    if $L_i > avgL$ **then**
        X_{index}=p;
    end if
    if $L_i \leqslant avgL$ **then**
        X_{index}=random(1,p-1) or X_{index}=random(p+1,m);
    end if
end for
return $R = (X, Y)$.

the makespan of the entire workflow will increase. If a task with a large amount of data is assigned to an instance with low processing capability, and the subsequent cross-mutation adopts a single-point cross-mutation method, the probability of the task being subsequently assigned to an instance with a high processor is not high. On the contrary, if the task is not reassigned to a processor with a high processing capacity, it will have a great impact on the completion time of the whole scheduling.

In order to reduce the impact of unreasonable initialization execution location on workflow's makespan, we consider the service length of tasks and then assign the different capacity instances to them. First, the data volume of each task needs to be evaluated. If the data volume of a task is greater than the average of volume of all tasks' data, the task is scheduled on VMs with a higher processing capacity. Otherwise, the task will be randomly arranged in other instances with a lower processing capacity.

Algorithm 1 gives the detailed steps of the chromosome initialization procedure, where S is used to store the current sortable tasks. t_{entry} must be scheduled at the beginning time, so

the first element of Y is t_{enrty}, which will be deleted from T(line 3). If all the $pred(t_i)$ of t_i are already in Y, the task t_i will be stored in S(lines 5-9). The next element of Y is randomly selected form S, and then T and S will be updated(lines 10-11). The initialization of Y will be finished util there is no element in T. To initialize the task execution position, X, the average of all tasks' service length and the max value of W should be determined(lines 13-14). Each task t_i in Y will be judged to executed in which instance according to the compare result of L_i and $avgL$ (lines 16-22). If a task's L_i is larger than $avgL$, the task will be scheduled in P_b, which has the largest execution capacity. Otherwise, the task will randomly select an instance to schedule.

3.2. Genetic operators

3.2.1. Crossover

The crossover operator can help to inherit some chromosome fragments of excellent individuals to future generations. In this study, when performing the crossover operator, individuals in the population are paired in pairs, and then a single point crossover method is adopted to generate new individuals.

According to the encoding strategy demonstrated by Fig. 2, the crossover is also divided into two parts: the crossover of task execution position and the crossover of task execution order. An example to demonstrate the crossover of task execution position is given in Fig. 3, there are two individuals in which represent the set of execution positions of the task $X_1 = \{x_1^1, x_1^2, ..., x_1^n\}$ and $X_2 = \{x_2^1, x_2^2, ..., x_2^n\}$. The process of obtaining new individual $X_1^{'}$ and $X_2^{'}$ after crossover operation is shown in Fig. 3. First, a randomly generated number within [1, n] is regarded as the crossover position. Then, a crossover operator is performed between X_1 and X_2 on the generated crossover position. Last, two new individuals of execution locations, i.e., $X_1^{'}$ and $X_2^{'}$, are obtained.

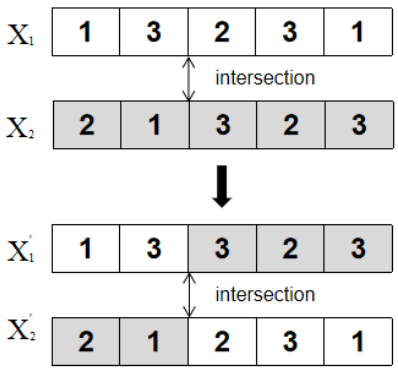


Fig. 3: Process of the single crossover of task execution position.

The sequence of task execution order also adopts the single point crossover operator. First, a randomly crossover position is selected. Then, the task sequence from the first task to the intersection in the sorting set is called the matching area. Suppose there are two tasks execution order sets Y_1 and Y_2, the matching area of set Y_1 is added to the front of

set Y_2, and the matching area of set Y_2 is added to the front of set Y_1 to form two new temporary individuals $\widetilde{y}_1$ and $\widetilde{y}_2$. Finally, the repetitive tasks in $\widetilde{y}_1$ and $\widetilde{y}_2$ are removed from the beginning to the end, and then the two remaining tasks are two new generated individuals denoting two task execution orders. In the process of removing duplicate tasks, because the task execution order in sets Y_1 and Y_2 has met the precedence constraints, the removal of tasks will not cause the conflict of task execution dependencies. The new individuals of two task execution order sets $Y_1 = \{t_1^1, t_1^2, ..., t_1^n\}$ and $Y_2 = \{t_2^1, t_2^2, ..., t_2^n\}$ after cross operation are $Y_1^{'}$ and $Y_2^{'}$, respectively. The process of the crossover is demonstrated by Fig. 4. Note that, since the task execution order in the sets Y_1 and Y_2 has met the precedence constraints, the generated individuals do not cause the conflict of task execution dependencies[46].

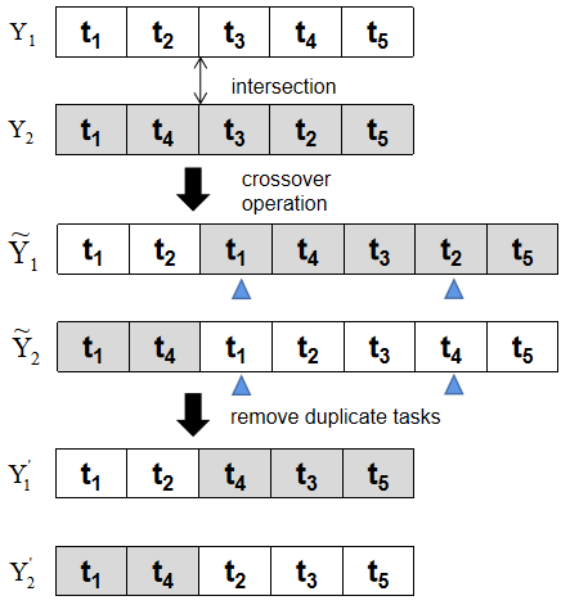


Fig. 4: Process of the single crossover of task execution order.

3.2.2. Mutation

Mutation operator in a GA refers to replacing some gene values with other alleles aiming to improving population diversity. Similar to the crossover operators detailed above, two mutate operators are performed on the task execution position and the execution order, respectively.

For the mutation of task execution position, a mutation point is randomly generated at first, and then a mutation probability pm is used to determine whether to mutate. If the mutation is determined, the value of this gene is mutated to another processor number. The mutation process of task execution position is shown in Fig. 5.

When conducting the mutation operator of the execution order, similar to the crossover operation of execution order, the preorder constraints between tasks also should to be con-

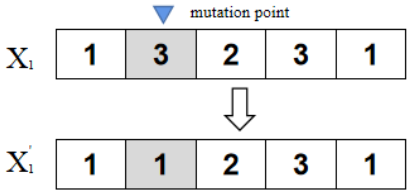


Fig. 5: Process of the mutation of task execution position.

sidered. In this study, firstly, we randomly select a gene y_i in Y to be mutated. Since the execution positions of the entry task t_{entry} and exit task t_{exit} are fixed, the task to be mutated cannot be the t_{entry} or t_{exit}. Then we search from the forward direction of Y. If all the $pred(y_i)$ being in set $\{y_1, y_2, .., y_a\}$ when we reach a gene y_a, the search is stopped. At the same time, we search from the reverse of set Y. If all the $succ(y_i)$ being in set $\{y_b, y_{b+1}., y_n\}$ when we reach a gene y_b, the search is stopped. At this time, task y_i must be in the set $Y' = \{y_{a+1}, ..., y_{b-1}\}$, and y_i can be in any position in the set Y'. Finally, task y_i excludes the original position and randomly selects a new position for insertion. The mutation process of task execution order is shown in Fig. 6.

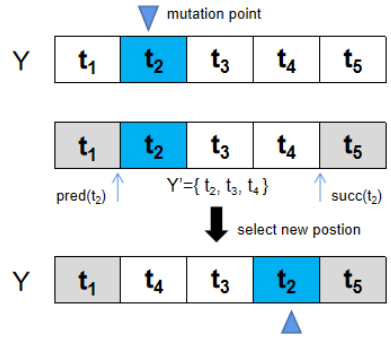


Fig. 6: Process of the mutation of task execution order.

3.2.3. Selection

Although various EAs applied in workflow scheduling problems offer promising performance, they have a common defect of high computational overhead. The main reason for the high computational overhead of this kind of algorithms is the low efficiency of individuals evaluation. How to reduce the computational overhead and make them better applied to QoS based multi-objective scheduling algorithms is our next main problem.

Individuals evaluation is essential in the random search algorithm based on evolutionary optimization. The fitness value of the population should be evaluated after each iterative update. In this study, we use the selection method combined with the LCS to save the same part of the first k optimal individuals in each generation to a set. Then, the crossover and mutation probability of individuals can be adjusted based on the LCS in the set, aiming to improve the search efficiency of the algorithm.

1) Longest Common Subsequence

A common subsequence of two or more strings is a subsequence that is common to these strings. Obviously, there are many common subsequences, in which the longest one is called as the LCS. An example of the LCS of S_1 and S_2 is illustrated in Fig. 7.

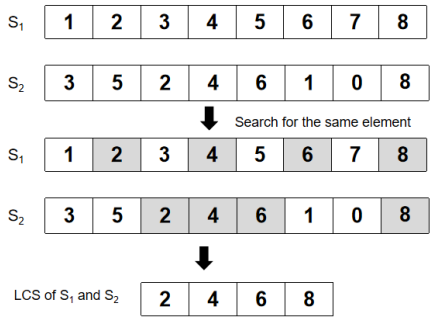


Fig. 7: An example of the LCS of two sequences.

Given two sequences $S_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $S_2 = \{3, 5, 2, 4, 6, 1, 0, 8\}$, there are many common subsequences of S_1 and S_2, in which the longest one is $\{2, 4, 6, 8\}$.

When using a dynamic programming algorithm to solve the LCS problem, we need to decompose the original problem into several subproblems, and then find the characteristics of LCS. Let $A = \{a_1, a_2, ..., a_m\}$, $B = \{b_1, b_2, ..., b_n\}$ be two sequences, and $C = \{c_1, c_2, ..., c_k\}$ be any of their common subsequences. After analysis, we can know:

1. If a_m=b_n, then a_m=b_n=c_k and $\{c_1, c_2, ..., c_{k-1}\}$ is an LCS of $\{a_1, a_2, ..., a_{m-1}\}$ and $\{b_1, b_2, ..., b_{n-1}\}$.
2. If a_m!=b_n and a_m!=c_k, then C is an LCS of $\{a_1, a_2, ..., a_{m-1}\}$ and B.
3. If a_m!=b_n and b_n!=c_k, then C is an LCS of A and $\{b_1, b_2, ..., b_{n-1}\}$.

Suppose we use $Z[i, j]$ to represent the length of LCS of A_i and B_j, where $A_i = \{a_1, a_2, ..., a_i\}$ and $B_j = \{b_1, b_2, ..., b_j\}$. The recursive formula is as follows:

$$Z[i,j] = \begin{cases} 0, & i = 0 \quad or \quad j = 0 \\ Z[i-1, j-1] + 1, & i, j > 0 \quad and \quad a_i = b_j \\ max(Z[i, j-1], Z[i-1, j]), & \\ & i, j > 0 \quad and \quad a_i \neq b_j \end{cases} \tag{16}$$

The pseudocode of getting two sequences' LCS is described in Algorithm 2, in which Z is used to record the length of LCS, while C is used to record the comparison results of each element of two sequences. First, we need to initialize Z (lines 3-8), then compare the two elements in the sequence according to (16), and update C and Z (lines 9-19).

The element of C calculated by Algorithm 2 can quickly obtain the LCS of sequences $A = \{a_1, a_2, ..., a_m\}$ and $B = \{b_1, b_2, ..., b_n\}$. When $C[i, j] = 1$, the LCS of A_i and B_j is the LCS of A_{i-1} and B_{j-1} plus a_i at the tail. When $C[i, j] = 2$, it means that the LCS of A_i and B_j is the same as that of A_i and B_{j-1}. When $C[i, j] = 3$, it means that the LCS of A_i

Algorithm 2 LCS-Length(A,B)

```
A = {a_1, a_2, ..., a_m}, B = {b_1, b_2, .., b_n};
let C = [1..m, 1...n] and Z = [1..m, 1...n] be new tables ;
for i = 1 to m do
    Z[i, 0]=0 ;
end for
for j = 0 to n  do
    Z[0, j]=0 ;
end for
for i = 1 to m do
    for j = 1 to n do
        if a_i==b_j then
            Z[i, j]=Z[i − 1, j − 1]+1, C[i, j]=1;
        else if Z[i − 1, j] ≥ Z[i, j − 1] then
            Z[i, j]=Z[i − 1, j], C[i, j]=2;
        else if Z[i, j]==Z[i, j − 1] then
            C[i, j]=3;
        end if
    end for
end for
return C and Z.
```

and B_j is the same as that of A_{i-1} and B_j. Starting from the last element of table C, we can quickly get the LCS.

2) Selection Combined LCS

The selection operation is a process to eliminate inferior individuals and retain superior individuals. At present, non dominated sorting and crowding distance sorting algorithms in NSGA-II [7] are some of the most widely used methods to retain elites in MOPs. Therefore, this study also uses the two methods to select individuals to generate offspring. First, the contemporary population and the new generation population are merged. Then a non dominated sorting method is carried out on the merged population. The individuals in each layer are sorted according to the dominant relationship between individuals in the population. The crowding distance is calculated for the individuals in each layer and sorted according to the distance. Finally, n individuals are selected as the new population according to the ranking of individuals. Concretely, according to the non dominated stratification and the calculation of crowding distance, each individual I will have two attributes: a non dominated value $rank(I)$ and a crowding distance $dis(I)$. When selecting a new species group to generate the next generation population, it is preferred to select some individuals who has low non dominated ranking levels. If two individuals belong to the same non dominated level, the individual with a larger crowding distance should be selected to generate offspring.

In the iterative process of population, we find that some top-ranked individuals may

Algorithm 3 Selection Combined LCS

Require: A new population P and the population size n; The number of k;

1: let $bestLCS$=[1..k] and $bestK$=[1..k] be new tables ;
2: **for** i=1 to k **do**
3: $bestLCS[i]$=ø $bestK[i]$=ø ;
4: **end for**
5: **for** $i = 1$ to k **do**
6: select best k individual to $bestK$;
7: **end for**
8: **for** i=1 to $k - 1$ **do**
9: L=getLCS($bestK(i)$,$bestK(i + 1)$), $bestLCS = bestLCS \bigcup L$;
10: **end for**
11: **for** i=1 to n **do**
12: **for** j=1 to k **do**
13: $L = getLCS$(P(i),$bestLCS(j)$);
14: **if** L.equals($bestLCS(j)$) **then**
15: reduce $P(i).pm$ and $P(i).pc$;
16: **end if**
17: **end for**
18: **end for**

have some common subsequences. Intuitively, these common subsequens may be favorable gene blocks, and preventing these gene blocks from being destroyed may be beneficial for speed up convergence. Thus, in order to prevent the favorable gene blocks when performing the crossover or mutation operations, we adopt the method combined with LCS selection to dynamically adjust the mutation and crossover probability of individuals in the newly generated population.

Algorithm 3 gives the detailed selection strategy combined with LCS. The $bestLCS$ is used to store the top k LCS, and $bestK$ is used to store the top k individuals. First, we need to initialize $bestLCS$ and $bestK$ (lines 2-4). According to the non dominated ranking mentioned above, the top-ranked individuals are selected as the population of the next generation. And then the top k individuals of the new population are stored in the optimal individual set, $bestK$ (lines 5-7). The individuals in the optimal individual set are paired to generate the LCS saved to the optimal LCS set, $bestLCS$ (lines 8-10). Each individual of the population, P is paired with the sequence in $bestLCS$ to generate LCS. If the generated LCS is the same as one of the sequences in $bestLCS$, the individual's probability of crossover and mutation is reduced (lines 11-18).

In order to solve the problem that LCS selection may fall into local optima, we consider delaying the selection of LCS. When GA uses the information obtained in the evolutionary process to organize the search by itself, individuals with better fitness have a higher survival probability and obtain a genetic structure that is more suitable for the environment. Therefore, GA has the characteristics of self-organization, self-adaptation and self-learning. At

the beginning of the iteration, we do not intervene in its selection. After a certain number of iterations, we add LCS selection strategy to accelerate the convergence of the algorithm. We use δ to represent the coefficient of delayed LCS selection. The parameter setting of δ will be explained in section 4.3.2.

Algorithm 4 GALCS()

Require: The DAG of workflow, the VM instances pool; population size $pNum$, iterations number $iNum$, the number of k and δ;

1: let *pop*, *result*, *newpop*, *oldpop* be new sets ;
2: **for** i=1 to $pNum$ **do**
3: generate new chromosomes by Algorithm 1, store the chromosomes to *pop*;
4: **end for**
5: $i = 0$
6: **while** $i < iNum$ **do**
7: use NSGA-II to sort *pop* and upadate *oldpop*;
8: **if** $i > \delta$ **then**
9: update *pm* and *pc* of each chromosome in *oldpop* by Algorithm 3;
10: **end if**
11: **for** $j = 1$ to $pNum$ **do**
12: p_1=random(0,1),p_2=random(0,1);
13: **if** $p_1 < oldpop[j].pc$ and $p_1 < oldpop[j+1].pc$ **then**
14: c_1, c_2=crossover($oldpop[j], oldpop[j+1]$);
15: **end if**
16: **if** $p_2 < oldpop[j].pm$ and $p_2 < oldpop[j+1].pm$ **then**
17: c_1=mutate($oldpop[j]$),c_2=mutate($oldpop[j+1]$);
18: **end if**
19: $newpop$=$newpop \cup c_1 \cup c_2$;
20: **end for**
21: *pop*=*oldpop*+*newpop*, $newpop = \emptyset$;
22: i++;
23: **end while**
24: use NSGA-II to sort *pop* and strore to *result*.

Ensure: *result*

3.3. Proposed algorithm

The proposed algorithm, named GALCS, is designed to retain some elite segment of a scheduling sequence. Algorithm 4 presents the detailed steps of GALCS. The algorithm works as follows: (1) The initial population, *pop*, is generated through Algorithm 1 (lines 2-4). (2) The non dominated sorting and crowding distance sorting algorithms in NSGA-II are used to generate the population, *oldpop*, which is to be crossover and mutate (line 7). (3) If the number of iterations is greater than δ, the selection combined LCS in Algorithm 3 will be used to update the chromesomes' *pm* and *pc*, where the chromesomes contain

bestLCS (lines 8-10). (4) Chromesomes in *oldpop* perform crossover and mutation to get new individuals (lines 11-20). (5) The *oldpop* and *newpop* are mixed together to update *pop* (line 21). (6) Repeat steps 2-5 until the number of iterations is reached and then the result calculated by NSGA-II is regarded as the final trade-off solution(line 24).

The source code of GALCS can be downloaded from:
https://github.com/XuewenXia-EvolutionaryComputation

4. Experimental results and discussions

In this section, comprehensive performance of GALCS, measured by 3 popular metrics, is testified by extensive experiments. Moreover, characteristics of new proposed strategies in GALCS are also verified by a set of experiments.

4.1. Comparison metrics

When measuring the performance of a multi-objective algorithm, we need to evaluate a solution from multiple aspects. In this study, Q-metric, FS-metric, and S-metric are selected as comparison metrics [18]. Definitions of the three metrics are detailed as follows.

1. Q-metric
   Q-metric is used to compare the convergence of solutions between two multi-objective algorithms [38]. Suppose that $PF_{\mathcal{A}_1}$ and $PF_{\mathcal{A}_2}$ are the Pareto front of the two comparison algorithms respectively, let $Y = PF_{\mathcal{A}_1} \cup PF_{\mathcal{A}_2}$, $\Phi = PF_{\mathcal{A}_1} \cap PF_{\mathcal{A}_2}$ and $\Omega = Y \cap PF_{\mathcal{A}_2}$. We define $Q(F_{\mathcal{A}_1}, F_{\mathcal{A}_2}) = |\Phi|/|Y|$, $Q(F_{\mathcal{A}_2}, F_{\mathcal{A}_1}) = |\Omega|/|Y|$, and the sum of $Q(F_{\mathcal{A}_1}, F_{\mathcal{A}_2})$ and $Q(F_{\mathcal{A}_2}, F_{\mathcal{A}_1})$ is 1. If $F_{\mathcal{A}_1}$ has a better convergence than $F_{\mathcal{A}_2}$if and only if:

$$Q(F_{\mathcal{A}_1}, F_{\mathcal{A}_2}) > Q(F_{\mathcal{A}_2}, F_{\mathcal{A}_1}) \quad or \quad Q(F_{\mathcal{A}_1}, F_{\mathcal{A}_2}) > 0.5 \tag{17}$$

2. FS-metric
   FS-metric is adopted to evaluate the diversity of Pareto optimal solutions found by an algorithm. The calculation of FS-metric is defined as Eq. (18):

$$FS = \sqrt{\sum_{i=1}^{m} \min_{(x_0,x_1)\in PF_{\mathcal{A}_1}\times PF_{\mathcal{A}_2}} (F_{\mathcal{A}_i}(x_0) - F_{\mathcal{A}_i}(x_1))^2} \tag{18}$$

   The greater the FS-metric measure is, the better diversity of the Pareto front has.
3. S-metric
   S-metric is applied to quantify the uniformity of an achieved Pareto front. The calculation of S-metric is defined as Eq. (19):

$$S = \sqrt{\frac{1}{N_p} \sum_{i=1}^{N_p} \left(d_i' - \bar{d}'\right)^2} \tag{19}$$

Table 1: Instance types based on Amazon EC2

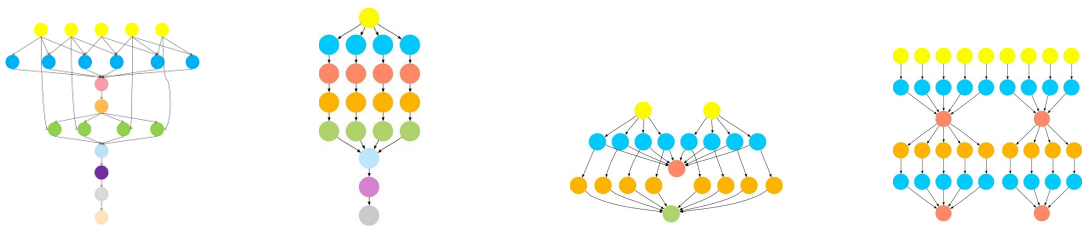
Type	vCPU	ECU	Memory (GB)	Energy (KW/h)
c5.large	0.5	2	4	0.085
m5.large	0.8	4	8	0.096
m5.xlarge	1	13	16	0.192
m5.2xlarge	2	20	32	0.384
m5.4xlarge	4	50	64	0.768
m5.12xlarge	8	130	128	2.304

where N_p is the number of Pareto optimal solutions, $\bar{d'} = \sum_{i=1}^{N_p} d_i'/N_p$, where d_i' represents the distance between a solution and its the nearest solution in the Pareto front. The smaller the value of S-metric is, the more uniform the Pareto optimal solution is.

4.2. Experimental settings

In this study, we use the CloudSim toolkit to simulate the cloud environment [5]. The experiments are based on the specifications and pricing scheme of Amazon EC2 [8], characteristics of which are presented in Table 1. The average bandwidth between two different instances is fixed as 20 Mbps. The value of the vCPU of each instance is considered as its processing capacity in Million Instruction Per Second (MIPS). The energy consumption of all VM types is assumed to be one hour.

Pegasus project has published the workflow of a number of real-world applications including Montage, CyberShake, Epigenomics, and LIGO Inspiral Analysis [4]. These workflows processing different characteristics have been widely used to measure the performance of workflow scheduling problems [15]. Montage is used to construct custom astronomical image mosaics of the sky. The CyberShake workflow is proposed by the Southern Calfornia Earthquake Center to characterize earthquake hazards in a region. The Epigenomics workflow is applied to map the epigenetic state of human cells on a genome-wide scale. LIGO's Inspiral Analysis workflow is introduced to generate and analyze gravitational waveforms from data collected during the coalescing of compact binary systems. The DAGs of the four different workflow structures are illustrated in Fig. 8.



(a) Montage (b) Epigenomics (c) CyberShake (d) Inspiral

Fig. 8: DAGs of four synthetic workflows.

4.3. Algorithm parameter discussions

4.3.1. Initialization method

In this paper, we propose an initialization method for the processor allocation based on tasks' data size. To testify the effectiveness of this scheme, we carry out some experiments in this part. Concretely, we use 3 kinds of VMs (i.e., c5.large, m5.large, and m5.xlarge) to execute 4 different sizes of workflows mentioned above. The comparison results, in terms of the average makespan of 10 runs, are demonstrated in Fig. 9, in which GA represents the random initialization being adopted while GA-2 denoting the method improved by us. It can be observed from Fig. 9 that our algorithm has a shorter makespan than the GA with the random initialization in both small-scale workflow and large-scale workflow.

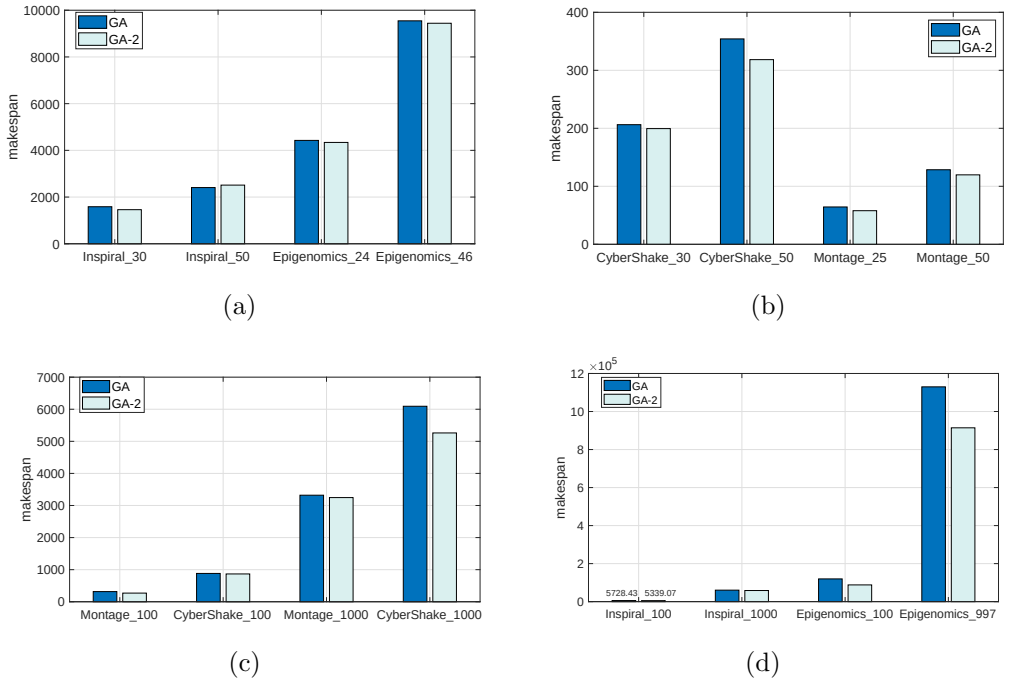
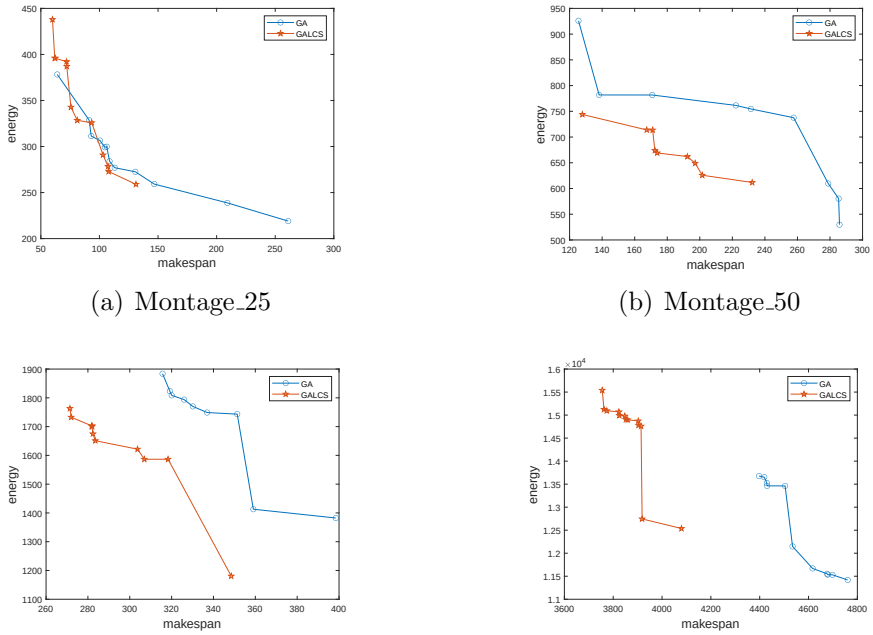


Fig. 9: Comparison of makespan between two kinds of initialized methods.

4.3.2. Comparison of different selection methods

In this study, the LCS-based selection is used to save some promising gene blocks, and then to speed up convergence. Thus, we also execute some experiments in this part to understand characteristics of the LCS-based selection method based on four different sizes of Montage. The comparison results measured by Pareto front between GA and GALCS are presented in Fig. 10. From the results we notice that introducing LCS in GA can significantly reduces the makespan. Meanwhile, it also can help GA to attain a better Pareto front.

GA mainly expands the search space of the population through crossover and mutation. However, if the probability of individual crossover and mutation is reduced in the early



(a) Montage_25 (b) Montage_50

(c) Montage_100 (d) Montage_1000

Fig. 10: Comparison of Pareto front between GA and GALCS.

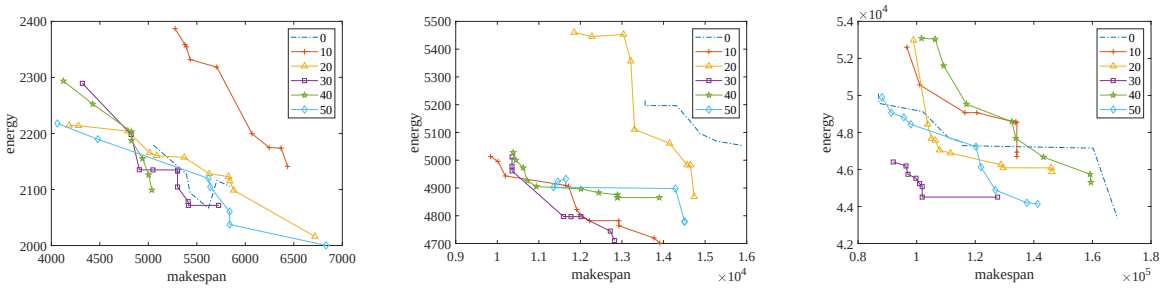
iterative process, it will reduce the possibility of the algorithm generating a better solution, which will easily lead to the algorithm falling into the local optimum. Therefore, we consider to delay the selection of LCS.

The parameter δ, which represents the coefficient of delayed LCS selection, is determined by the following experiments. In order to investigate how the parameters δ affects the performance of GALCS, we compared 6 different δ values. The experimental results are shown in Fig. 11, from which we can see that the delayed LCS can obtain a better Pareto solution set. Concretely, when the scale of the problem is small, GA can get a better solution quickly. Therefore, delaying LCS in the small-scale workflow has little impact on problem-solving. In the large-scale workflow, delaying LCS selection can get a better Pareto front, especially when δ is set as 30. Thus, in this study, δ is set as 30 when the number of iterations is 100.

4.4. Compared algorithm

To verify the performance of GALCS, we select 5 popular algorithms, i.e., LACO [42], SPSO [27], MOHEFT [9], GAHEFT [6] and LAGA [37], as competitors to compare the Pareto front. What's more, we add two existing simple heuristic based algorithms, i.e., CEDA [13] and FMProx [11], as additional peer algorithms to compare the makespan and energy consumption, respectively.

MOHEFT, as an extension of HEFT, builds a solution by iteratively mapping tasks onto instances. First, the priority of each task needs to be calculated. Then it maps the task into instances through the priority of the task. MOHEFT mapping all the tasks to be executed



(a) Epigenomics_24 (b) Epigenomics_46 (c) Epigenomics_100

Fig. 11: Impact of the number of coefficient δ.

onto all possible instances so as to extend the number of solutions. Finally, it employs the crowding distance to measure the solutions and select the best k individuals, who have greater crowding distances, to form the Prato font.

SPSO adopts PSO to address the optimization problem which aims to minimize the overall execution cost while meeting a user-defined dead-line constraints. In SPSO, tasks are randomly mapped onto VM instances by a particle without considering the impact of the tasks' order. It also considers fundamental features of the dynamic provisioning and heterogeneity of unlimited computing resources as well as VM performance variation.

In LACO, an ACO is employed to address the workflow scheduling problem which needs to meet dead-line constraints while minimizing cost. It takes a probabilistic list scheduling metric to distribute the whole deadline to each of the tasks. Then it uses the proposed probabilistic upward rank to order tasks. It uses different ant to consider different task scheduling sequences.

GAHEFT is a hybrid algorithm based on HEFT and GA. It generates an initial scheduling sequence using HEFT and then uses it as a part of the initial population of the GA. This schema can greatly improve the convergence of the algorithm, though it slightly reduces its searching ability.

LAGA determines the task order of solutions by using a maxmin strategy. It proposes the reliability-driven reputation, which is used to effectively evaluate the reliability of a resource in widely distributed systems.

CEDA is proposed for the workflow scheduling in order to optimize the total execution time and economic cost while meeting the deadline constraints. CEDA selects a task which has the highest upward rank and dispatches the selected task to the slowest instance of VMs.

FMProx uses RoundRobin method to address a workflow scheduling. Each workflow application is partitioned into sub-workflows in order to minimize data dependencies among them. The sub-workflows are executed on instance selected based on the QoS constraints. Finally, the sub-workflows can be executed on each instance to minimize the partial makespan.

4.5. Results and discussions

4.5.1. Comparsion of makespan-energy trade-offs

This section compares the results of workflows with different tasks executed by the GALCS, SPSO, LACO, MOHEFT, GAHEFT and LAGA. First, we plot the produced makespan-energy trade-offs for different sizes of Montage, Epigenomics, and Inspiral workflow in Fig. 12. It can be seen that with the increase of the number of tasks, the makespan of workflow and the energy consumption of processors increase. In majority of the workflows, the Pareto optimal solution obtained by GALCS is the best, which reflects the superiority of GA in solving multi-objective problems.

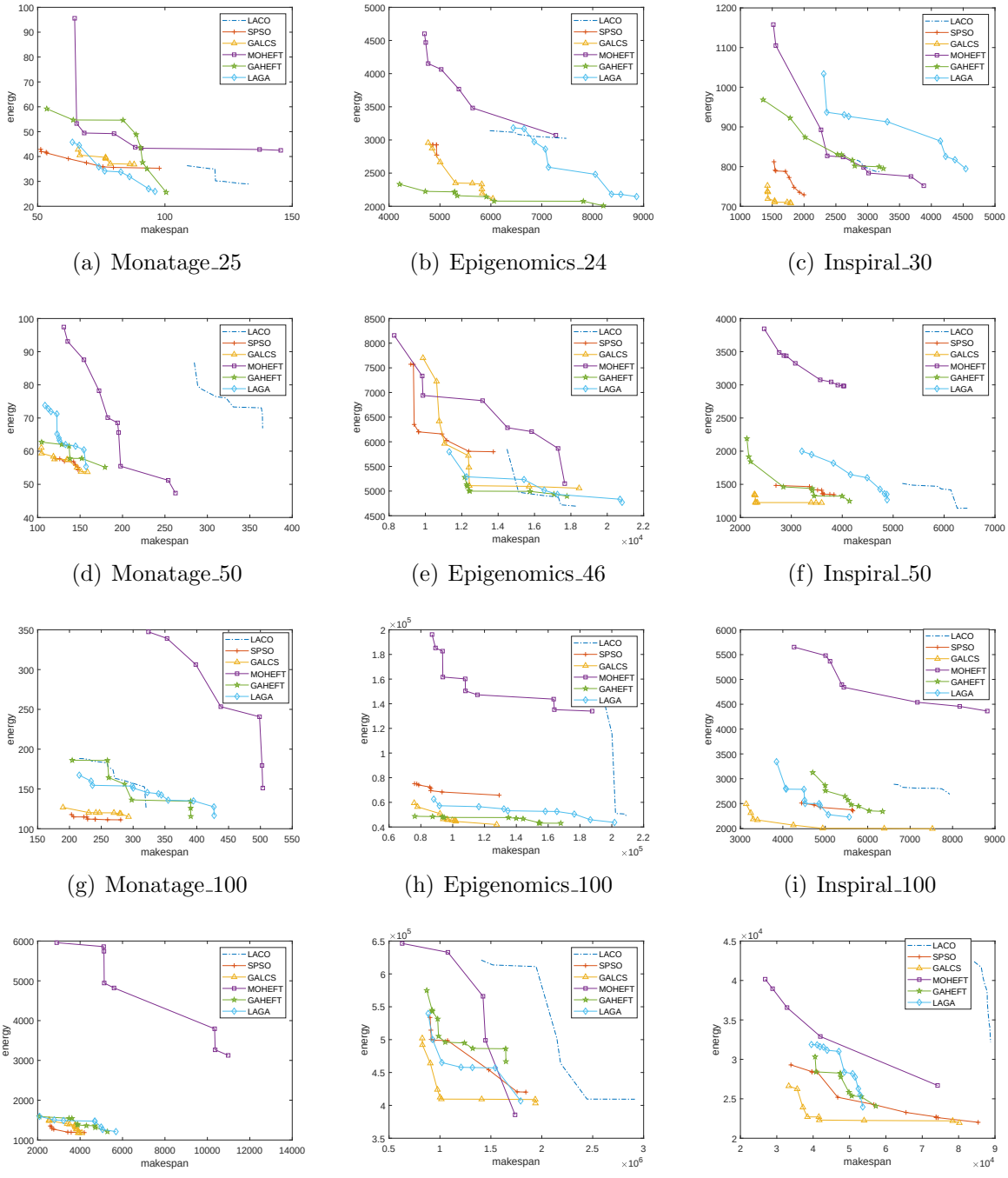
Among the small size workflow, the Pareto front results on Montage_25 scheduled by GALCS and SPSO are better than MOHEFT and LACO respectively in energy consumption and makespan. Although GALCS is slightly worsen than SPSO and GAHEFT, in terms of Parato front, GALCS attains the most promising performance on Inspiral_30 measured by Parato front. Concretely, GALCS can yield solutions with the lowest energy consumption than other peer algorithms. Furthermore, it also can offer a solution with almost the lowest makespan. On Inspiral_30, GALCS not only achieves the most favorable Parato front, it also can search some solutions with the lowest energy consumption or the lowest makespan, followed by SPSO.

Among the medium size workflow, GALCS and SPSO also manifest very promising characteristics. For example, GALCS and SPSO achieve almost the same performance, in terms of the energy consumption, while GALCS attains some solutions with the shortest makespan. On Inspiral_50, GALCS not only can obtain solutions with the best performance on the energy consumption, but also can achieve solutions with the best characteristics on the makespan. On Epigenomics_46, the 6 algorithms offer their own distinct properties. For instance, LACO can obtain solutions with the lowest energy consumption though it displays unfavorable performance on makespan. On the contrary, MOHEFT yields the best performance, in terms of makespan, though it does not attain very promising performance measured by energy consumption. Comparing the performance between GALCS and SPSO we can observe that the former displays more outstanding performance on the energy consumption while the latter offers more favorable property on the makespan.

For the large size workflow problems, GALCS dominates other 5 competitors on Inspiral_100 and Epigenomics_100 except GAHEFT, in terms of the Pareto front. In other words, GALCS can obtain solutions with the lowest energy consumption or the shortest makespan. On Montage_100, GALCS is slightly worsen than SPSO, in terms of the energy consumption, while GALCS can find out solutions with shorter makespan than SPSO. Although LACO yields mediocre performance on Montage_100, it offers unfavorable properties on Epigenomics_100 and Inspiral_100.

Among the extra-large size workflow problems, all the 6 peer algorithms display almost the same properties as that among the large size workflow problems. In other words, GALCS can exhibit more favorable and comprehensive performance than other peer algorithms.

In general, GALCS can achieve a better trade-off between makespan and energy consumption among the 6 algorithms, followed by SPSO, GAHEFT, LAGA, LACO, and MOHEFT. Concretely, MOHEFT can quickly find a feasible solution for each workflow, but



(a) Monatage_25 (b) Epigenomics_24 (c) Inspiral_30

(d) Monatage_50 (e) Epigenomics_46 (f) Inspiral_50

(g) Monatage_100 (h) Epigenomics_100 (i) Inspiral_100

(j) Monatage_1000 (k) Epigenomics_997 (l) Inspiral_1000

Fig. 12: Comparsion of four algorithms in terms of Pareto front for differnt workflows.

the solution quality is unstable. The reason may be that MOHEFT allocates processors according to a fixed task scheduling sequence, so the solution space is limited. Moreover, MOHEFT only determines the priority according to the restriction relationship between the

EFT of the task, so it performs poorly in terms of energy consumption. On the contrary, in the most workflow execution results, the distribution of GALCS is compact. The reason is that GALCS uses the LCS selection strategy to select the next generation of population, which makes the majority of the final results are distributed near the optimal solution selected by LCS. Although the LCS selection can bring some merits for GA, it may cause GA to fall into local optimization. It also can be observed that SPSO yields very promising properties in some workflow schedule problems.

4.5.2. Performance evaluation

In this section, the Q-metric, FS-metric, and S-metric are used to analyze the characteristics of the 6 peer algorithms based on the experimental results presented in Section 4.5.1.

Table 2: Q-metric comparison of different workflows

Q-metric		LACO	SPSO	MOHEFT	GAHEFT	LAGA
Montage_25	GALCS	F	F	T	T	F
Montage_50	GALCS	T	T	T	T	T
Montage_100	GALCS	T	T	T	T	T
Montage_1000	GALCS	F	T	T	T	T
Epigenomics_24	GALCS	F	T	T	F	T
Epigenomics_46	GALCS	F	T	T	F	T
Epigenomics_100	GALCS	F	T	T	T	T
Epigenomics_997	GALCS	F	T	T	T	T
Insprial_30	GALCS	F	T	T	T	T
Insprial_50	GALCS	F	T	T	T	T
Insprial_100	GALCS	F	T	T	T	T
Insprial_1000	GALCS	F	T	T	T	T

Table 3: FS-metric comparison of different workflows

	FS-metric					
	LACO	SPSO	MOHEFT	GAHEFT	LAGA	GALCS
Montage_25	0.19	0.33	**0.83**	0.79	0.64	0.21
Montage_50	1.47	0.17	**2.01**	1.06	1.37	0.11
Montage_100	2.29	0.51	**8.35**	0.14	1.44	1.79
Montage_1000	0.77	16.18	**100.76**	0.05	1.52	1.52
Epigenomics_24	4.26	1.05	**91.78**	2.38	5.77	5.78
Epigenomics_46	2.35	1.7	**87.1**	23.43	23.21	23.21
Epigenomics_100	52.36	212.19	**1220.948**	107.38	158.85	201.38
Epigenomics_997	1246.17	1578.9	**27805.41**	1964.56	3416.38	3417.72
Inspiral_30	2.45	9.8	**39.52**	36.87	9.8	0.12
Inspiral_50	3.48	12.086	**16.1**	4.7	4.15	0.12
Inspiral_100	25.45	30.18	**70.17**	12.07	16.34	57.29
Inspiral_1000	64.16	44.3	100.76	**182.54**	126.95	1.52
CyberShake_30	0.15	1.5	**7.15**	0.87	0.84	0.09
CyberShake_50	0.43	0.77	5.57	**5.61**	0.76	0.48
CyberShake_100	10.26	13.02	11.19	0.89	1.23	**17.87**
CyberShake_1000	21.17	34.28	**178.13**	44.68	13.02	22.84

A. Q-metric comparison

As described in Section 4.1, Q-metric obtained by a pairwise comparison is used to measure the convergence performance between two algorithms.

In Table 2, if an algorithm has a better convergence than another algorithm, the Q-metric is T, otherwise, the value is F. The results in Table 2 indicate that GALCS dominates SPSO, LAGA and MOHEFT measured by convergence speed on most test workflows except on

Table 4: S-metric comparison of different workflows

	S-metric					
	LACO	SPSO	MOHEFT	GAHEFT	LAGA	GALCS
Montage_25	2.57	5.31	12.34	2.47	2.64	**0.96**
Montage_50	35.49	2.28	4.77	3.03	4.35	**2.24**
Montage_100	**6.03**	7.21	15.17	14.16	8.79	11.94
Montage_1000	**38.29**	58.13	623.76	426.92	56.85	38.93
Epigenomics_24	172.3	**1.61**	505.88	2.38	93.29	90.91
Epigenomics_46	255.44	482.11	473.76	372.09	**31.32**	450.03
Epigenomics_100	5263.087	11310.798	7819.134	8113.67	5162.30	**4987.94**
Epigenomics_997	107771.24	120247.3	153627.93	92672.40	119810.29	**84497.58**
Inspiral_30	19.69	26.29	65.14	133.56	119.17	**13.30**
Inspiral_50	3356.747	183.7405	128.5663	159.47	121.89	**41.82**
Inspiral_100	125.27	**100.21**	344.2	150.65	149.56	419.56
Inspiral_1000	38.93	52.13	623.76	112.487	344.26	**38.29**
CyberShake_30	5.34	8.58	26.36	18.62	16.08	**2.17**
CyberShake_50	8.59	49.79	89.45	14.2	44.69	**7.84**
CyberShake_100	115.1	84.38	621.13416	68.53	98.91	**66.81**
CyberShake_1000	591.3	844.84	9138.93	656.38	645.93	**477.21**

Montage_25, where SPSO and LAGA performs better than GALCS. However, the convergence of GALCS is lower than LACO in the most workflow. This is due to LACO employing ACO to carry out the optimization. The positive feedback mechanism adopted in it is beneficial for speeding up the convergence. However, from Fig. 11 we can see that GALCS can obtain a better Pareto front than LACO. The comparison results verify that the more uniform individual distribution in GALCS is beneficial for the population diversity. Thus, GALCS has a greater probability of obtaining the global optimal solution is, though optimization time will be longer. On the contrary, the more concentrated individuals in LACO cause a lower population diversity, which is not conducive to the exploration ability of the algorithm, though it accelerates the convergence speed of LACO.

B. FS-metric comparison

FS-metric is used to evaluate the diversity of the Pareto front. The greater value of FS-metric is, the better performance of the algorithm is.

It can be observed from Table 3 that MOHEFT attains the most favorable results on almost all the problems except GALCS displays the best performance on CyberShake_100 and GAHEFT performs best on Inspiral_1000 and CyberShake_50. The favorable performance of MOHEFT algorithm relies on that the algorithm exhausts the mapping scheme of the current task at each time, and then uses the crowding distance to select the optimal task sequence. On the contrary, GALCS accelerates the optimization process by combining the GA algorithm of LCS. Therefore, the solution of GALCS is dominated by MOHEFT, in terms of the diversity of the Pareto front.

C. S-metric comparison

S-metric is used to measure the uniformity of the Pareto front. The comparison results of S-metric are presented in Table 4, in which the smaller value of the S-metric is, the better uniform the Pareto optimal solution is.

On all test workflows, the S-metric value of GALCS is the best accounting for 68.8% on all workflows, followed by SPSO and LACO, who achieve the best S-metric value on 2 workflow scheduling problems. Although MOHEF offers the best performance, in terms of the FS-metric, it cannot yield the best result on any problem. From the favorable characteristics

of GALCS we can obtain a preliminary conclusion that the adopted the selection strategy combined with LCS can make the population evolve towards the established goal, and then the solution distribution is more uniform.

4.5.3. Makespan and energy consumption evaluation

Table 5: Comparison of makespan for all workflow applications

workflow	scheduling algorithm							
	CEDA	FMprox	GAHEFT	LAGA	LACO	SPSO	MOHEFT	GALCS
Montage-25	61.64	61.64	53.58	63.65	108.67	**51.20**	64.53	65.90
Montage-50	108.90	108.54	**104.31**	108.66	284.59	121.06	130.79	104.67
Montage-100	219.64	220.21	204.17	215.39	215.38	203.15	324.08	**189.84**
Montage-1000	2100.58	2097.52	**2079.80**	2089.59	10806.63	2595.05	2897.17	2538.26
Epigenomics_24	8921.80	8921.80	**4207.48**	6440.06	5978.83	4857.57	4696.14	4768.39
Epigenomics_46	10002.25	11167.36	12158.93	11309.02	14489.43	9196.77	**8285.96**	9867.26
Epigenomics_100	86201.91	83571.15	76206.10	88094.95	186405.59	76044.84	87038.57	**75813.97**
Epigenomics_997	893986.76	875522.62	870405.52	884851.70	1405677.11	902233.50	**629455.28**	826162.49
CyberShake_30	216.06	208.45	190.34	204.57	310.36	219.02	**140.85**	217.32
CyberShake_50	320.76	353.10	325.15	311.35	702.27	352.50	**253.46**	308.60
CyberShake_100	635.32	696.49	635.39	619.95	663.29	671.04	866.91	**556.92**
CyberShake_1000	3965.17	3998.03	3984.56	3963.84	4009.04	4081.72	4796.49	**3955.02**
Inspiral_30	1711.32	1801.65	**1356.77**	2308.44	2828.24	1526.91	1517.37	1425.44
Inspiral_50	2664.07	2574.91	**2126.92**	3208.34	5180.16	2698.66	2466.89	2275.79
Inspiral_100	3770.60	3948.86	4703.58	3850.61	6614.39	4441.12	4265.20	**3135.92**
Inspiral_1000	39919.34	40129.36	40539.09	39515.26	84265.64	33896.66	**26818.24**	32019.75

To examine separately the performance on makespan and energy consumption of each algorithm, a set of experiments is conducted in this section, results of which are described in Table 5 and Table 6. It can be seen from Table 5 that GALCS performs significantly better than CEDA, FMprox, LACO and LAGA on almost all the real-world workflows. Also, GALCS performs better than GAHEFT on most of the Epigenomics and CyberShake workflows except on Epigenomics_24 and CyberShake_30. And it's easy to observe that GALCS can obtain the shortest makespan in majority of the large size workflows. GALCS can achieve a shorter makespan on most workflows due to it has the advantage of both GA and LCS.

It is evident from table 6 that the amount of energy consumption is slower in GALCS as

Table 6: Comparison of energy consumption for all workflow applications

workflow	scheduling algorithm							
	CEDA	FMprox	GAHEFT	LAGA	LACO	SPSO	MOHEFT	GALCS
Montage-25	25.99	25.99	**25.63**	25.96	28.93	35.28	42.49	36.84
Montage-50	55.40	55.37	55.17	55.38	66.89	54.35	**47.36**	53.72
Montage-100	116.63	116.68	115.51	116.71	124.84	121.21	151.14	**115.09**
Montage-1000	1211.25	1211.08	1210.05	1210.73	4350.20	1186.57	3127.05	**1178.31**
Epigenomics_24	2339.70	2339.70	**2007.78**	2146.11	3024.16	2771.08	3071.74	2114.97
Epigenomics_46	**4603.06**	4645.44	4896.91	4771.65	4694.42	5797.98	5151.72	5058.56
Epigenomics_100	43971.63	43616.05	43273.96	43702.64	48932.12	65764.54	134003.68	**41965.36**
Epigenomics_997	408108.70	406311.82	406095.18	406788.28	409369.23	420180.27	499407.83	**403216.91**
CyberShake_30	98.24	97.75	96.68	97.19	97.66	98.03	157.40	**92.27**
CyberShake_50	**180.87**	183.43	182.27	181.32	198.91	206.56	335.80	185.87
CyberShake_100	375.24	382.26	374.96	373.66	912.63	**368.65**	2780.19	372.08
CyberShake_1000	2413.19	2416.47	2415.49	2413.67	2647.85	2650.94	2879.82	**2409.29**
Inspiral_30	742.83	745.52	721.39	794.20	787.39	728.88	751.75	**712.12**
Inspiral_50	1295.60	1277.84	1247.42	1266.56	**1139.46**	1344.25	2981.72	1236.15
Inspiral_100	2230.47	2238.09	2342.25	2228.28	2684.91	2352.97	4363.00	**2000.67**
Inspiral_1000	24007.82	24022.53	24082.42	23968.70	38218.63	22017.45	26691.35	**21592.74**

compared to FMprox and LAGA for almost all workflows. Moreover, GALCS also performs better than CEDA, GAHEFT, LACO, SPSO and MOHEFT for all the workflows in large size workflows. That is, as the size of the workflow increases, the energy consumption of GALCS is better than that of other algorithms.

5. Conclusion and future work

This paper proposes a GA combined with the LCS selection (GALCS) to deal with cloud workflow scheduling problems considering makespan and energy consumption simultaneously. Firstly, we develop an initial method for allocating processors based on the amount of task data and processing capacities of VMs. Furthermore, we find that some of the task scheduling subsequences have a positive impact on the execution results of workflow. Thus, to reduce the destructive effect of crossover and mutation on these subsequences, we use a selection algorithm combined with LCS to maintain the integrity of this part of the subsequence. In simulation experiments with real-world data, the algorithm compares with other state-of-art algorithms under the same environment. The results show that GALCS can obtain a better Pareto optimal solution than other algorithms, with good convergence and uniformity. The results also verify that the performance of selection combined with LCS is competitive.

The performance of the algorithm proposed in this paper has been confirmed in simulation experiments, but there is still room for improvement. For example, there will be problems of premature convergence and insufficient universality of results in the algorithm. Therefore, how to prevent premature convergence and improve the universality of algorithm results by using GALCS is one of the future works. In addition, there are many idle time slots in the scheduling process. How to use idle time slots to improve the task completion time, is also a very practical technology.

Acknowledgment

The study was funded by the National Natural Science Foundation of China (Grant No.: 61663009), the Natural Science Foundation of Fujian Province (Grant Nos.: 2021J011008, 2021J011007), the Science and Technology Plan Projects of Zhangzhou (Grant Nos: ZZ2020J06, ZZ2020J24), and the Natural Science Foundation of Education Department of Jiangxi Province (Grant No.: GJJ200629).

References

[1] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682-694.

[2] M. Armbrust, A. Fox, R. Griffith, et al, Above the clouds: a berkeley view of cloud computing, Commun. of ACM. 53 (4) (2009) 50-58.

[3] H. Aziza, S. Krichen, A hybrid genetic algorithm for scientific workflow scheduling in cloud environment, Neural Comput. Appl. 32 (2020) 15263-15278.

[4] S. Bharathi, A. Chervenak, E. Deelman, et al, Characterization of scientific workflows, in: Proceedings of Third Workshop on Workflows in Support of Large-Scale Science, 2008, pp. 1-10.

[5] R.N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, et al, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Soft. Pract. Exp. 41 (1) (2010) 23-50.
[6] A.M. Chirkin, A.S.Z. Belloum, SV. Kovalchuk, et al, Execution time estimation for workflow scheduling, Future Gener. Comput. Syst. 75 (2017) 376-378.
[7] K. Deb, S. Agrawal, A. Pratap, et al, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-II, in: Proceedings of International Conference on Parallel Problem Solving from Nature, 2000, pp. 849-858.
[8] J.J Durillo, R. Prodan, Multi-objective workflow scheduling in amazon ec2, Cluster Comput. 17 (2) (2013) 169-189.
[9] J.J. Durillo, H.M. Fard, R. Prodan, Moheft: A multi-objective list-based method for workflow scheduling, in: Proceedings of 4th IEEE International Conference on Cloud Computing Technology and Science, 2012, pp. 185-192.
[10] R. Garg, M. Mittal, L.H. Son, Reliability and energy efficient workflow scheduling in cloud environment, Cluster Comput. 22 (2019) 1283-1297.
[11] T. Ghafarian, B. Javadi , R. Buyya, Decentralized workflow scheduling in volunteer computing systems, Int. J. Parallel Emergent. Distrib. Syst. 30 (5) (2015) 343-365.
[12] A. Gupta, H.S. Bhadauria, A. Singh, Load balancing based hyper heuristic algorithm for cloud task scheduling, J. Ambient. Intell. Humaniz. Comput. 12 (2020) 5845-5852.
[13] R.A. Haidri, CP. Katti , PC. Saxen , Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing, J. King Saud University - Comput. Inf. Sci. 32 (2017) 666-683.
[14] A. Jaszkiewicz, Genetic local search for multi-objective combinatorial optimization, Eur. J. Oper. Res. 137 (1) (2002)50-71.
[15] G. Juve, A. Chervenak, E. Deelma, et al, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2012)682-692.
[16] S. Katoch, SS. Chauhan, V. Kumar, A review on genetic algorithm: past, present, and future, Multimed. Tools. Appl. 80 (5) (2020) 8091-8126.
[17] Y.C. Lee, H. Han, A.Y. Zomaya, et al, Resource-efficient workflow scheduling in clouds, Konwl. Based Syst. 80 (2015) 153-162.
[18] H. Li, Q. Zhang, Multiobjective optimization problems with complicated pareto sets, MOEAD and NSGA-II, IEEE Trans. Evol. Comput. 13 (2) (2009) 284-302.
[19] M. Li, Research on the mechanism and influence factors of urban style building based on cloud computing logistics information, Cluster Comput. 22 (6) (2019) 13873-13880.
[20] M. Li, S. Yang, K. Li, X. Liu, Evolutionary algorithms with segment-based search for multiobjective optimization problems, IEEE Trans. Cyber. 44(2014) 1295-1313.
[21] M. Li, S. Yang, X. Liu, Shift-based density estimation for pareto-based algorithms in many-objective optimization, IEEE Trans. Evol. Comput. 18 (3) (2014)348-365.
[22] Z.Y. Li, S.M. Chen, B. Yang, et al, Multi-objective memetic algorithm for task scheduling on heterogeneous cloud, Chinese J. Comput. 39 (2) (2016) 377-390.
[23] J. Peng, M. Liu, X. Zhang, et al, Hybrid heuristic algorithm for multi-objective scheduling problem, J. Syst. Eng. Electron. 30 (2) (2019) 327-342.
[24] F. Ramezani, J. Lu, J. Taheri, et al, Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments, World Wide Web. 18 (6) (2015) 1737-1757.
[25] F. Ramezani, M. Naderpour, J. Taheri, et al, Task scheduling in cloud environments: a survey of population based evolutionary algorithms, Evol. Comput. Sched. (2020) 213-225.
[26] S. Ran, A model for web service discovery with QoS, ACM SIGecom Exchanges. 4 (1) (2003) 1-10.
[27] M.A. Rodriguez, R. Buyya, Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds, IEEE Trans. Cloud Comput. 2 (2) (2014) 222-235.
[28] H. Singh, S. Tyagi, P. Kumar, et al, Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions, Simul. Model. Pract.

Theory. 111 (2021) 102353.
[29] V. Singh, I. Gupta, P. K. Jana, An Energy Efficient Algorithm for Workflow Scheduling in IaaS Cloud, J. Grid Comput. 18 (2019) 357-376.
[30] H. Topcuoglu, S. Hariri, W. Min-You, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel. Distrib. Syst. 13 (3) (2002) 260-274.
[31] D.H. Tran, J.S. Chou, D.L. Long, Optimizing non-unit repetitive project resource and scheduling by evolutionary algorithms. Oper. Res. 22 (2020) 77-103.
[32] G. Tremblay, R. Sabourin, P. Maupin, Optimizing nearest neighbour in random subspaces using a multi-objective genetic algorithm, in: Proceedings of the 17th International Conference on Pattern Recognition, 2004, pp. 208-211.
[33] J.D. Ullman, Np-complete scheduling problems, J. Comput. Syst. Sci. 10 (3) (1975)384-393.
[34] B. Varghese, R. Buyya, Next generation cloud computing: New trends and research directions, Future Gener. Comput. Syst. 79 (2018) 849-861.
[35] A. Verma, S. Kaushal, A hybrid multi-objective particle swarm optimization for scientific workflow scheduling, Parallel Comput. 62 (2017) 1-19.
[36] J. Wang, D. Li, Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing, Sens. 19 (5) (2019) 1023.
[37] X. Wang, CS. Yeo, R. Buyya, et al, Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm, Future Gener. Comput. Syst. 27 (8) (2011) 1124-1134.
[38] Y. Wang, C. Dang, Improving multiobjective evolutionary algorithm by adaptive fitness and space division, Adv. Nat. Comput. 2005, pp. 392-398.
[39] Y. Wang, X. Zuo, An effective cloud workflow scheduling approach combining pso and idle time slot-aware rules, IEEE/CAA J. Automatic. Sinica. 8 (5) (2021) 1079-1094.
[40] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, J. Supercomp. 71 (9) (2015) 3373-3418.
[41] L. Wu, S. Garg, R. Buyya, SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments, in: Proceedings of 11th IEEE ACM Int. Symp. Clust. Cloud Grid Comput., Cloud and Grid Computing. 2011, pp. 195-204.
[42] Q. Wu, F. Ishikawa, Q. Zhu, et al, Deadline-constrained cost optimization approaches for workflow scheduling in clouds, IEEE Trans. Parallel. Distrib. Syst. 28 (12) (2017) 3401-3412.
[43] H.I. Yung, F. Paas, Effects of cueing by a pedagogical agent in an instructional animation: A cognitive load approach, J. Educ. Techno. Soc. 18 (3) (2015) 153-160.
[44] Y. Zhang, Research on the security mechanism of cloud computing service model, Automat. Contr. Comput. Sci. 50 (2) (2015) 98-106.
[45] X. Zhou, G. Zhang, T. Wang, et al, Makespan-Cost-Reliability Optimized Workflow Scheduling using Evolutionary Techniques in Clouds, J. Circuits Syst. Comput. 29 (10) (2019) 2050167.
[46] Y.M. Zhou, Z.J. Li, J.D. Ge, et al, Multi-objective workflow scheduling based on delay transmission in mobile cloud computing, J. Softw. 29 (11) (2018) 3306-3325.
[47] Z. Zhu, G. Zhang, M. Li, et al, Evolutionary multi-objective workflow scheduling in cloud, IEEE Trans. Parallel. Distrib. Syst. 27 (5) (2016) 1344-1357.
[48] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE Trans. Evol. Comput. 3 (4) (1999) 257-271.