Received: 4 April 2016 | Revised: 22 July 2016 | Accepted: 22 July 2016

DOI 10.1002/cpe.3942

WILEY

RESEARCH ARTICLE

Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing

Li Liu[1] | Miao Zhang[2,1*] | Rajkumar Buyya[3] | Qi Fan[1]

[1] School of Automation and Electrical Engineering, University of Science and Technology, Beijing, China

[2] School of Information and Electronics, Beijing Institute of Technology, Beijing, China

[3] The University of Melbourne, Parkville, VIC, Australia

Correspondence
Miao Zhang, School of Information and Electronics, Beijing Institute of Technology, Beijing, China.
Email: s20130926@ustb.edu.cn

Summary

The cloud infrastructures provide a suitable environment for the execution of large-scale scientific workflow application. However, it raises new challenges to efficiently allocate resources for the workflow application and also to meet the user's quality of service requirements. In this paper, we propose an adaptive penalty function for the strict constraints compared with other genetic algorithms. Moreover, the coevolution approach is used to adjust the crossover and mutation probability, which is able to accelerate the convergence and prevent the prematurity. We also compare our algorithm with baselines such as Random, particle swarm optimization, Heterogeneous Earliest Finish Time, and genetic algorithm in a WorkflowSim simulator on 4 representative scientific workflows. The results show that it performs better than the other state-of-the-art algorithms in the criterion of both the deadline-constraint meeting probability and the total execution cost.

KEYWORDS

cloud computing, coevolutionary genetic algorithm, resource scheduling, scientific workflow

1 | INTRODUCTION

Scientific experiments are usually represented as workflows,[1] where tasks are linked according to their data flow and compute dependencies. Such scientific workflows are data-intensive and compute-intensive applications; for example, Compact Muon Solenoid experiment for the Large Hadron Collider at CERN[2] produces a huge amount of data to be analyzed, which are more than 5 peta-bytes per year when running at peak performance. The Human Genome Project is aimed at sequencing and identifying all 3 billion chemical units in the human genetic instruction set and discovering the genetic roots of disease to find treatments. These scientific workflows have tremendous data and computing requirements and need a high-performance computing environment for execution. Cloud computing is the latest development of distributed computing, grid computing, and parallel computing,[3,4] which delivers the dynamically scaling computing resources as a utility, much like how water and electricity were delivered to households these days. The patterns that cloud computing provides resources contain the following: Infrastructure as a Service (IaaS), Platform as a Service, and Software as a Service.[3,5] In this paper, we refer to IaaS cloud, which offers us a virtual pool to provide unlimited virtual machines (VMs).

The main character of cloud computing is virtualization. Cloud enables to provide computational resources in the form of VMs. A process that maps tasks in a workflow to compute resources (VMs) for execution (preserving dependencies between tasks) is called workflow scheduling. There are 2 layers for cloud workflow scheduling, which include VM-task mapping and the execution order for tasks in a single VM. In this paper, we just use the evolutionary approach to schedule VM-task mapping in the first layer, and the execution order in a single VM is set as that in paper,[6] where the VM will schedule the task with the smallest end time. There are different optimization objectives for workflow scheduling in cloud, including makespan, cost, throughput, and load balancing. In this paper, the optimization objective is the cost of executing a scientific workflow and subject to a deadline constraint, which is to find a proper task-VM mapping strategy, which minimizes the total financial cost and the makespan satisfies the deadline constraint.

The context above is a constrained optimization problem, and to transform constrained problem into unconstrained one, most evolutionary algorithms usually use static penalty function to penalize infeasible solutions by reducing their fitness values in proportion to the degrees of constraint violation. However, it is difficult to set a suitable penalty factor. Another common method is to eliminate the infeasible individuals within their evolutionary process. However, some infeasible individuals usually hold very rare and excellent genes that are very valuable for the next generations and are unable to be eliminated. We propose a coevolutionary genetic algorithm (CGA) with adaptive penalty function for the constrained scientific workflow scheduling in clouds. We have considered the main features of cloud providers such as heterogeneous computing resources and dynamic provision. An adaptive penalty function is applied in the CGA, which will adjust itself automatically during the evolution. And we apply the notion of coevolution to adjust the crossover and mutation probability factors, which are helpful for the convergence. Our main contributions can be summarized as follows: (1) present the optimization model of scientific workflow scheduling in cloud environment, which is cost minimization and deadline constrained, and consider cloud resources' dynamic provision pattern and heterogenetic character. (2) Propose a new CGA with adaptive penalty function approach CGA^2, for deadline-constrained scientific workflow scheduling in clouds. It applies a self-adaptive penalty function into the CGA, which is able to efficiently prevent prematurity and uses the notion of coevolution to adjust the crossover and mutation probabilities, which can efficiently accelerate the convergence. (3) Unlike existing genetic approaches, we generate the initial population based on the critical path (CP),[7] which can also efficiently prevent prematurity and improve deadline meeting for workflow scheduling. Simulation results demonstrate that our approach has high accuracy in terms of deadline-constraint satisfaction at lower costs.

The remainder of this paper is organized as follows: the following section introduces the related works. The context of our model is presented in Section 3. Section 4 presents the CGA^2 approach and the adaptive penalty function. Section 5 applies the proposed approach (CGA^2) to cloud scientific workflow scheduling problem. Section 6 evaluates the performance of the CGA^2 and has made comparison with the existing algorithms and then gives the experimental results. We conclude the paper with a discussion and a description of future work in Section 6.1.

2 | RELATED WORK

To address the problem of constrained workflow scheduling in cloud computing, we have adopted some evolutionary algorithms to generate near-optimal solutions. Wang and Yeo et al[8] presented a look-ahead GA, which used the reliability-driven reputation to evaluate the resource's reliability. Moreover, the multiobjective model aiming to optimize both the makespan and the reliability of a workflow application was proposed. A particle swarm optimization (PSO)-based approach was proposed in 1 study,[9] which aimed to minimize the execution cost of a workflow while balancing the task load on the available resources. Rodriguez and Buyya[6] presented a cost-minimization and deadline-constrained PSO approach for cloud scientific workflow scheduling. It considered fundamental features of IaaS providers, like providing elastic and heterogeneous computing resources (VMs). A penalty function was used in their algorithms that the particles violating the constraints are inferior to the feasible particles. However, this method would lead a premature convergence, which is very common in the PSO algorithms.

Sawant[10] used the GA for VMs configuration in cloud computing. He incorporated the constraints into the objective fitness function, so as to transform the constrained optimization problem to an unconstrained one. This is the most popular way to handle the constrained optimization problem and is particularly easy to implement. But it is very hard to set a suitable penalty factor to tradeoff between the global optimization searching and the constraints satisfaction. The coefficients they used have no physical meaning and were obtained through empirical evaluation. Huang[11] proposed a new improved GA, where the chromosomes not only represent the computing resource-task assignment but also indicate the queue on the VMs' task being executed. It first evolves individuals according to the optimization objective and changes the evolved population based on the constrained objective when the individuals violate the constraint. This approach releases the burden of devising an appropriate penalty function for constrained optimization problem. However, it needs to evolve for numerous generations, and a feasible solution may not be found.

The ant colony optimization approach was used for VMs configuration in cloud computing, which aimed to be energy efficient.[12] The experiments showed that the proposed approach achieved superior energy gains through better server use and required less resource than the first-fit decreasing approach. A PSO for workflow scheduling in cloud computing was proposed in 1 study,[13] which considered both the computation cost and the data transmission cost. The experiments showed that the PSO can achieve as much as 3 times of cost savings, and the workload is better than that of the existing best resource selection algorithm.

3 | PROBLEM FORMULATION

A workflow is depicted as $G = (V, E)$, where $V = \{t_1, t_2, \dots, t_n\}$ and E is the vertices and edges of the graph, respectively. Each vertex represents a task t, and there are n tasks in the workflow. The edges maintain execution precedence constraints. Having a directed edge $e_{x,y}$ from t_x to t_y, $x, y \in M$ means that t_y cannot start to execute until t_x is completed; task t_x is a parent task of t_y, and t_y is a child task of t_x. Tasks without parents are called the entry task t_{entry}, and tasks without child tasks are called the exit task t_{exit}. Each workflow has a deadline d_W associated to what determines the allowed longest time to complete its execution. Figure 1 shows an example of workflow, in which each node represents a task and the arcs show the data transfer between nodes.

The IaaS cloud providers offer a range of VM types denoted by $\overline{VM} = VM_1, VM_2, \dots, VM_n$. Different VM types provide different computing resources, and then we define VM type in terms of its processing capacity P_{VMi} and cost per unit of time C_{VMi}. Virtual machines are charged per unit of time τ, if $\tau = 60$ minutes, use of VM for 61 minutes

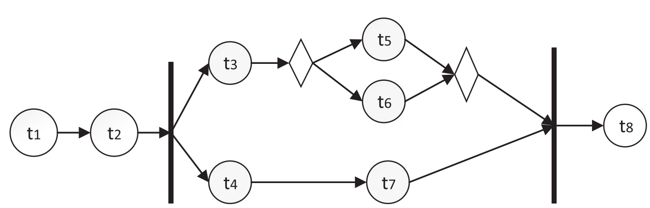

FIGURE 1 Workflow example

would incur a payment of 2 hours (2 units of time). We assume that each task is executed by a single VM, and a VM can execute several tasks.

The running time $RT_{ti}^{VM_{ti}}$ of task t_i executed by VM_{ti} is calculated in Equation 1, where s_{ti} is the size of task t_i and P_{VMti} is the processing capacity of VM_{ti}. The transfer time $TT_{ei,j}$ between a parent task t_i and its child task t_j is depicted in Equation 2, where $d_{t_i}^{out}$ is the output data size produced by task t_i, β is the bandwidth between each VM, and the bandwidth for all VMs is roughly same. If 2 tasks are executed in the same VM, the transfer time is 0.

$$RT_{t_i}^{VM_{t_i}} = s_{t_i}/P_{VM_{t_i}} \tag{1}$$

$$TT_{e_{i,j}} = d_{t_i}^{out}/\beta \tag{2}$$

There are many different optimization objectives for workflow scheduling in clouds. In this paper, we focus on finding the optimization solution for workflow scheduling, which can minimize the total execution cost and satisfy the deadline constraint. We define a scheduling vector $S = (M, TEC, TET)$ in terms of tasks to resources matching M, the total execution cost TEC, and the total execution time TET. M is the task-VMs matching, which is composed of VM types, start time, and end time for all tasks, $M = \left(m_{t1}^{VM_{t1}}, m_{t2}^{VM_{t2}}, \ldots, m_{tM}^{VM_{tM}}\right)$, $m_{t_i}^{VM_{t_i}} = (t_i, VM_{ti}, ST_{ti}, ET_{ti})$, which means task t_i is associated with VM_{ti}, and the start time and the end time for task t_i are calculated by the following equations:

$$ST_{t_i} = \begin{cases} LET_{VM_{t_i}}, & \text{if } t_i \text{ is an entry task} \\ \max\left(\max\limits_{t_a \in parent(t_i)} (ET_{t_a} + TT_{e_{a,i}}), LET_{VM_{t_i}}\right) & \text{otherwise} \end{cases}, \tag{3}$$

$$ET_{t_i} = ST_{t_i} + RT_{t_i}^{VM_{t_i}}, \tag{4}$$

where LET_{VMti} is the lease end time of VM_{ti}, which is also the time when VM_{ti} becomes idle. There will be no charge for data transfers within the same data center, and we do not consider this fee when calculating the workflow total cost. The total execution cost TEC and the total execution time TET are calculated as in the following equations:

$$TEC = \sum_{i=1}^{|VM|} C_{VM_{t_i}} \times \left\lceil \frac{RT_{t_i}^{VM_{t_i}}}{\tau} \right\rceil + \sum_{i \in T, j \in T} TT_{e_{i,j}} \times TC_{VM_{t_i}}, \tag{5}$$

$$TET = \max(ET_{t_i} : t_i \in V), \tag{6}$$

where C_{VMti} is the processing cost for VM_{ti}, and TC_{VMti} is the data transfer cost for VM_{ti}.

The problem in this paper can be described as finding a scheme S with the minimum TEC, and the TET does not exceed the workflow's deadline constraint d_W, as shown in the following equation.

$$\begin{aligned} &\text{Minimize } TEC \\ &\text{Subject to}: \ TET \le d_W \end{aligned} \tag{7}$$

In this paper, we use a CGA with adaptive penalty function approach (CGA^2) to solve this optimization problem. Table 1 gives the description of notations used in our paper.

TABLE 1 Notations

Notation	Description
VM_i	Virtual machine i
P_{VMi}	Processing capacity for VM_i
C_{VMi}	Cost per unit of time for VM_i
RT_{ti}^{VMi}	Running time of task t_i executed by VM_i
s_{ti}	The size of task t_i
$TT_{ei,j}$	The transfer time between a parent task t_i and child task t_j
$d_{t_i}^{out}$	The output data size of task t_i
β	The bandwidth between each VM
TEC	The total execution cost
TET	The total execution time
d_W	The workflow's deadline constraint
ST_{ti}	The starting time of task t_i
ET_{ti}	The ending time of task t_i
LET_{VMti}	The lease ending time of VM_{ti}, also the time that VM_{ti} is idle

4 | COEVOLUTIONARY GA WITH ADAPTIVE PENALTY FUNCTION

In this section, we have described a CGA with adaptive penalty function approach (CGA^2) to solve constrained optimization problem.

4.1 | Mechanism of the CGA

Coevolution is the process of mutual adaptation of two or more populations. The key issue in the coevolutionary algorithms is that the evolution of a population depends on another population.

In 1994, Paredis[14] introduced CGA. Coello[15] incorporated the notion of coevolution into a GA to adapt genetic coefficients and to solve constrained optimization problems. In this work, we will use the notion of coevolution to adjust the crossover and mutation probabilities and apply an adaptive penalty function coefficients scheme in the GA.

The structure of coevolution model in CGA^2 is shown in Figure 2. In CGA^2, 2 types of populations are used. In particular, 1 type of a single population (denoted by $Population_2$) with size M_2 is used to adapt suitable crossover and mutation probability factors, and another kind of multiple populations (denoted by $Population_{1,1}$, $Population_{1,2}$,..., $Population_{1,M2}$) in that each of them is with size M_1 that evolves in parallel with different crossover and mutation schemes to search good

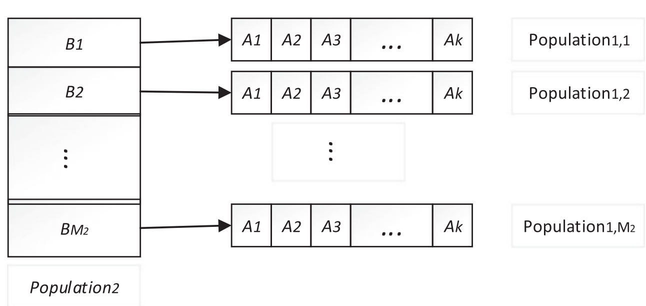

FIGURE 2 Coevolution model

decision solutions. Each individual B_j in $Population_2$ represents a set of crossover and mutation probability factors for individuals in $Population_{1,j}$, where each individual represents a decision solution.

In every generation of coevolution process, each $Population_{1,j}$ will evolve by the GA for a certain number of generations (G_1) with crossover and mutation probability factors obtained from individual B_j in $Population_2$. Then the fitness of each individual B_j in $Population_2$ will be determined. After all individuals in $Population_2$ are evaluated, $Population_2$ will also evolve by using GA. The above coevolution process will repeat until a predefined stopping criterion is satisfied (eg, a maximum number of coevolution generation G_2 is reached).

In short, 2 types of populations evolve interactively, where $Population_{1,j}$ with an adaptive penalty function scheme is used to evolve decision solutions, while $Population_2$ is used to adapt crossover and mutation probabilities for solution evaluation. Owing to the coevolution, not only decision solutions are explored evolutionary but also crossover and mutation probabilities are adjusted in a self-tuning way to avoid the difficulty of setting suitable factors by trial and error.[16]

4.2 | Adaptive crossover and mutation probabilities

In GA, the bigger the crossover probability p_x and mutation probability p_m are, the more new individuals will be generated along with the diversity of population. But if the probabilities are too big, good genes will be destroyed easily; otherwise, if the probabilities are too small, it is not conducive to generate new individuals, and the search speed will slow down.[17]

According to Zhang et al,[18] the evolutionary process in GA can be depicted as 4 states, including initial state, submaturing state, maturing state, and matured state. The crossover probability p_x and mutation probability p_m are adjusted differently based on population states. For example, if there are almost inferior individuals with extremely poor fitness, we should increase p_m and decrease p_x, like in the initial state.

If the span of fitness values in a population is very large, the crossover probability p_x should be increased, like submaturing state and maturing state. If there are almost excellent individual with good fitness, we could decrease p_x and p_m as in the matured state. According to these rules and inspired by the idea in the literature,[19,20] a self-adaptive crossover and mutation operator is described as in the following equations:

$$p_x(i) = \omega_1 \times \cos\left(\frac{\pi}{2} \times \frac{1}{e^{(\sigma_1(i) + \sigma_2(i) + \cdots + \sigma_m(i))}}\right), \tag{8}$$

$$p_m(i) = \omega_2 \times f_i / f_p, \tag{9}$$

where $p_x(i)$ is the crossover probability for ith individual and $p_m(i)$ is the mutation probability. $\sigma_m(i) = |f_i - f_i(m)|/f_i$, $f_i(m)$ is the mth closest individual to individual i in terms of fitness value. f_i is the fitness value of individual i, and f_p is the individual with biggest fitness value in the population. ω_1 and ω_2 are the crossover and mutation probability factors, which are adapted through coevolution.

4.3 | Fitness calculation with adaptive penalty function

A suitable penalty function plays an important role in the performance of GAs, and it becomes more important when the constraints are stricter, which means that the optimal solution is nearing the boundary between the feasible and infeasible search space.[21,22] The most common form of the penalized fitness function is depicted as follows:

$$F_i^a = F^a(x_i) = F(x_i) - P(x_i) = F(x_i) + \textstyle\sum_{j=1}^{m} \lambda(j) \times E_j(x_i), \tag{10}$$

where F_i^a is the fitness function of ith individual after penalty. $F(x_i)$ is the fitness value for the optimizing objective, $\lambda(j)$ is the penalty factor for the jth constraints violation, m is the number of constraints, and $E_j(x_i)$ is the jth constraint violation for ith individual.

Inspired by Nanakorn et al,[23] the penalty function should be suitable for all infeasible individuals. If it is too small, many infeasible individuals may have higher penalized fitness value, and the population would move toward a false direction to the infeasible region. Otherwise, if it is too large, some individuals with good genes will be eliminated, which may lead to premature convergence. According to Tessema et al,[24] an adaptive penalty function strategy is applied to keep track of the number of feasible individuals in the population to determine the amount of penalty added to the infeasible individuals.

First, each individual's fitness value and constraint violation will be normalized by Equations 11 and 12.

$$\widetilde{F}(x_i) = \frac{F(x_i) - F_{\min}}{F_{\max} - F_{\min}}, \tag{11}$$

where $\widetilde{F}(x_i)$ is the normalized fitness value, and $F_{\min}$ and $F_{\max}$ are the smallest and largest fitness values for all individuals in the current population. In this way, each individual's fitness will lie between 0 and 1. And the normalized constraint violation $\widetilde{E}(x_i)$ of each infeasible individual is calculated as follows:

$$\tilde{E}(x_i) = \frac{1}{m} \textstyle\sum_{j=1}^{m} \frac{E_j(x_i)}{E_j^{\max}}, \tag{12}$$

where m is the number of constraints and $E_j^{\max}$ is the maximum violation for jth constraints for all infeasible individuals.

Furthermore, for different evolutionary states in GA, the penalty rule should be adapted accordingly. For example, if there are few feasible individuals in the population, the infeasible individuals with lower constraint violation will be less penalized than those with higher constraint violation. On the other hand, if there are many feasible individuals in the population, the infeasible individual with lower normalized fitness should be less penalized. According to literature,[21,24] the final fitness value is formulated in the following.

1. If there is at least 1 feasible individual in the current population, the fitness function is as shown in the following equation:

$$F1^a(x_i) = \begin{cases} \widetilde{F}(x_i) & \text{for feasible individual} \\ \sqrt{\widetilde{F}(x_i)^2 + \widetilde{E}(x_i)^2} + \left[(1-r_f)\widetilde{E}(x_i) + r_f \widetilde{F}(x_i)\right] & \text{otherwise,} \end{cases} \tag{13}$$

where $r_f = \frac{number\ \ of\ \ feasible\ \ individuals}{population\ \ size}$. In this way, the individuals with both low fitness value and low constraint violation will be considered better than those with high fitness value or high constraint violation. And if the feasibility ratio (r_f) in the population is small, then the individual that is closer to the feasible space will be considered better. Otherwise, the individual with lower normalized fitness value will be better.

1. If there is no feasible individual in the current population, the fitness function is calculated as in the following equation.

$$F1^a(x_i) = \widetilde{E}(x_i) \tag{14}$$

Obviously, the individuals with smaller constraints violation are considered better. Consequently, the search will move to the region where the sum of constraints violation is small (ie, the boundary of the feasible region).

The fitness value for ith individual in $Population_{1,j}$ in CGA^2 is evaluated based on Equations 13 and 14.

Each individual in $Population_2$ represents a set of factors (ω_1 and ω_2). After $Population_{1,j}$ evolves for G_1 generations, the jth individual B_j in $Population_2$ is evaluated by the following equation:

$$F2(B_j) = -\min(F1_j) + \frac{numinfeasible}{M_1} \tag{15}$$

where $F1_j$ is the fitness values for all individuals in $Population_{1,j}$, *numinfeasible* is the number of infeasible individuals in $Population_{1,j}$, and M_1 is the size of $Population_{1,j}$.

5 | THE CGA^2 FOR WORKFLOW SCHEDULING

5.1 | CGA^2 modeling

In this paper, we use 2 types of chromosomes to model CGA^2 for scientific workflow scheduling in clouds. As shown in Figure 3, $chremosome_{i,j}$ in $Population_{1,j}$ represents the decision solution, which is also the ordered pair of task-resource matching of a workflow. For the scheduling scenario here, the position of each gene in $chromosome_{i,j}$ is the task number, and the value of each gene in $chromosome_{i,j}$ is the VMs' number; thus, the dimension of a $chromosome_{i,j}$ is the number of tasks in a workflow. The range of genes in $chromosome_{i,j}$ is determined by the number of resource available to run the tasks. Figure 3 represents a workflow with 8 tasks and 5 VMs available. The fitness function is used to determine how good a decision solution is, which is calculated by the optimizing objective total execution cost *TEC* and the constraint total execution time *TET*. The calculation of *TEC* and *TET* for a chromosome is explained in the next section.

$chromosome_m$ in $Population_2$ represents the crossover and mutation probability coefficients, which is defined by the binary encoding. The range of ω_1 is (0, 1], and we use the first 7 genes to represent the coefficient ω_2. The value of ω_1 in $chromosome_m$ is calculated as $\omega_1 = \frac{2^6+2^4+2^2+1}{128} = 0.6640625$. The range of ω_2 is also (0, 1], and we use the latter 7 genes to represent the coefficient ω_2. And the value of ω_2 is calculated as $\omega_2 = \frac{2^5+2^3+1}{128} = 0.3203125$.

$Population_{1,j}$ is the evolution decision solutions to match the task with resource to minimize the total execution cost *TEC* and satisfy the constraint total execution time *TET*. $Population_2$ adapts the crossover and mutation probabilities for solution evolution.

5.2 | *TEC* and *TET* calculation

To address the workflow scheduling problem, we need to estimate the running time of workflow application with a specific task-VM mapping schedule first and calculate the cost accordingly. The total execution cost *TEC* and the total execution time *TET* of a $chromosome_{i,j}$ in $Population_{1,j}$ are shown in Algorithm 1. The kth position value of $chromosome_{i,j}(k)$ represents that task k is associated with $VM_{chromosomei,j(k)}$. In $Population_{1,j}$, a chromosome is a task-resource match.

First, we initialize the VMs state matrix *VS* and the task state matrix *TS*. A set of workflow tasks T and a set of VMs $\overline{VM}$ are inputted. Then we estimate the execution time $RT_{ti}^{VM_{ti}}$ of each workflow task $t_i (t_i \in T)$ on every type of VM $VM_i (VM_i \in \overline{VM})$ according to Equation 1, and transfer time $TT_{ei,j}$ between tasks is calculated according to Equation 2.

The starting time value ST_{ti} has 2 cases. If the task has no parent tasks, it can start as soon as the VM assigned to the task is idle. Otherwise, the task starts after the parent tasks finished and the output data are transferred. Furthermore, if the VM is still busy, the starting time has to be delayed until the VM is enabled. And in our algorithm, if 2 tasks allocated on the same VM have the same start time, the VM will process the task with smaller size. The ending time value ET_{ti} is

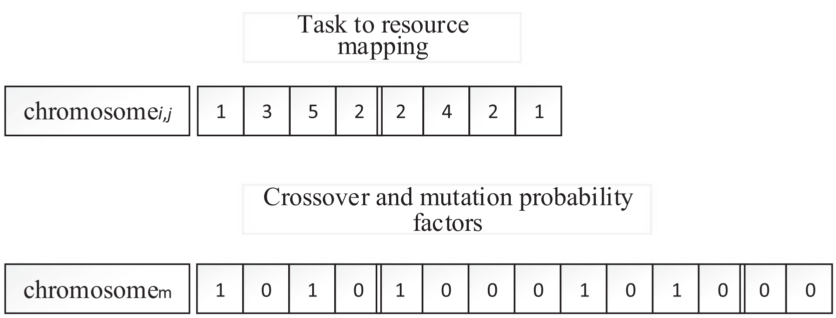

FIGURE 3 Chromosome encoding in the coevolution

calculated by Equation 4 based on the starting time and execution time $RT_{ti}^{VM_{ti}}$. After a task has been scheduled, we need to update the VS and the TS to set the task t_i as scheduled and the time period between ST_{ti} and ET_{ti} as busy for VM_{ti}. The process continues until all tasks having been scheduled.

Algorithm 1 *TEC* and *TET* Estimation

Input: *a set of workflow tasks T, a set of VMs* $\overline{VM}$, *and a chromosome*$_{k,j}$ *in populationion*$_{1,j}$

Output: *TEC and TET*

1. *Initialize* VMs state matrix *VS and* task state *matrix TS*.

2. *Calculate* execution time RT[$|T| \times |\overline{VM}|$]; *Calculate* transfer time TT[$|T| \times |T|$];

3. For i=1:$|T|$

  If $TS(T(i))$ is unscheduled

  3.1. $t_i = T(i)$, $VM_{ti} = vm_{chromosomek,j(i)}$

  3.2. If t_i *has no parents*

   $ST_{ti} = LET_{VMti}$;

  Else

$$ST_{ti} = \max\left(\max_{t_a \in parent(t_i)} (ET_{ta} + TT_{ea,i}), LET_{VMti}\right);$$

  End

3.3. For *each child task* t_c *of* t_i

  If t_c *is mapped to a VM different to* VM_{ti}

$$TT(i) = TT(i) + TT(i,c)$$

  End

  End

3.4. $RT_{ti}^{VM_{ti}} = RT(t_i, VM_{ti})$;

3.5. $ET_{ti} = RT_{ti}^{VM_{ti}} + TT(i)$

3.6. *update VS, and TS, set the time period* $[ST_{ti}, ET_{ti}]$ *for* VM_{ti} *is busy, set* $TS(T(i))$ *as scheduled.*

  End

End

4. *Calculate TEC according Equation 5;*

5. *Calculate TET according Equation 6;*

5.3 | Initial population

For a scientific workflow, the execution time of the tasks in the CP makes more influence on the total execution time of a workflow, while the financial execution cost of these tasks is a small part of the total execution cost. So allocating these tasks to the high-performance VMs will decrease the total execution time greatly while having just a little impact on the total financial execution cost.

The diversity of initial population impacts the performance of GA greatly, but most GAs generate the initial population randomly. To improve the solution quality and convergence speed, we generate one-fifth of the initial population based on CP[7] and assign these tasks in CP to the VMs with high processing capacity. The tasks in the other one-fifth of the initial population are allocated to the VMs, which have the lowest price. The rest of the population is produced in random.

Figure 4 shows the framework of CGA^2. We first initialize 2 types of populations, where $Population_2$ is used to adapt crossover and mutation probabilities for $Population_1$ to find decision solutions. In this paper, the evolution of $Population_2$ is an unconstrained optimization problem, which does need penalty function, while $Population_1$ uses an adaptive penalty function to transform the constrained workflow scheduling problem as an unconstrained optimization one. The initial population scheme used in $Population_1$ is depicted in Section 5.3, and $Population_2$ adopts the random method. Each subpopulation $Population_{1,j}$ in $Population_1$ will evolve for G_1 iterations simultaneously, and the best M_2 individuals from the M_2 subpopulations will be used to assess the corresponding individual in $Population_2$. $Population_2$ will evolve for G_2 iterations to find the best crossover and mutation probability factor and the best decision-making solution.

6 | PERFORMANCE EVALUATION

To evaluate the performance of CGA^2 in addressing the problem of scientific workflow scheduling in clouds, we use the WorkflowSim framework supported by CloudSim to simulate a cloud environment. The simulated workflows are 4 famous scientific workflows—Epigenomics, Montage, Inspiral, and CyberShake[25,26]—which are widely applied for performance measurement of scheduling algorithms in the WorkflowSim.[27,28] Each of these workflows has different structures as seen in Figure 5.[10]

We use related approaches for constrained optimization problem, such as the Random, Heterogeneous Earliest Finish Time (HEFT),[29] the GA,[10] and the PSO for deadline-constrained cloud scientific workflow scheduling,[9] as a baseline to evaluate our approach.

The Random is an algorithm that assigns the ready tasks to an idle VM randomly. The HEFT is a scheduling algorithm that gives higher priority to the workflow task, which has a higher rank value. This rank value is calculated by using average execution time for each task and average communication time between resources of 2 successive tasks, where the tasks in the CP have higher rank values. Then, it sorts the tasks in a decreasing order of their rank values, and the task with a higher rank value is given higher priority. In the resource selection phase, tasks are scheduled in the order of their priorities, and each task is assigned to the resource that can complete the task at the earliest time. We set $|T_x|$ as the size of task T_x and R as the set of resources (VMs) available with average processing power $|R| = \sum_{i=1}^{n} {|R_i|}/{n}$, and the average execution time of the task is defined as $E(T_x) = \frac{|T_x|}{|R|}$.

Let T_{xy} be the size of data to be transferred between task T_x and T_y and β be the bandwidth between each VM. Thus, the average data transfer time for the task is defined as $TT_{x,y} = T_{xy}/\beta$. $E(T_x)$ and $TT_{x,y}$ are used to calculate the rank of a task. Rank value is calculated as follows:

$$rank(T_x) = \begin{cases} E(T_x)T_x & \text{is an exit task} \\ E(T_x) + \max_{T_y \in child(T_x)} (TT_{x,y} + rank(T_y)) & \text{otherwise} \end{cases}. \quad (16)$$

A workflow is represented as a directed acyclic graph, and the rank values of the tasks in HEFT are calculated by traversing the task graph in a breadth-first search manner in the reverse direction of task

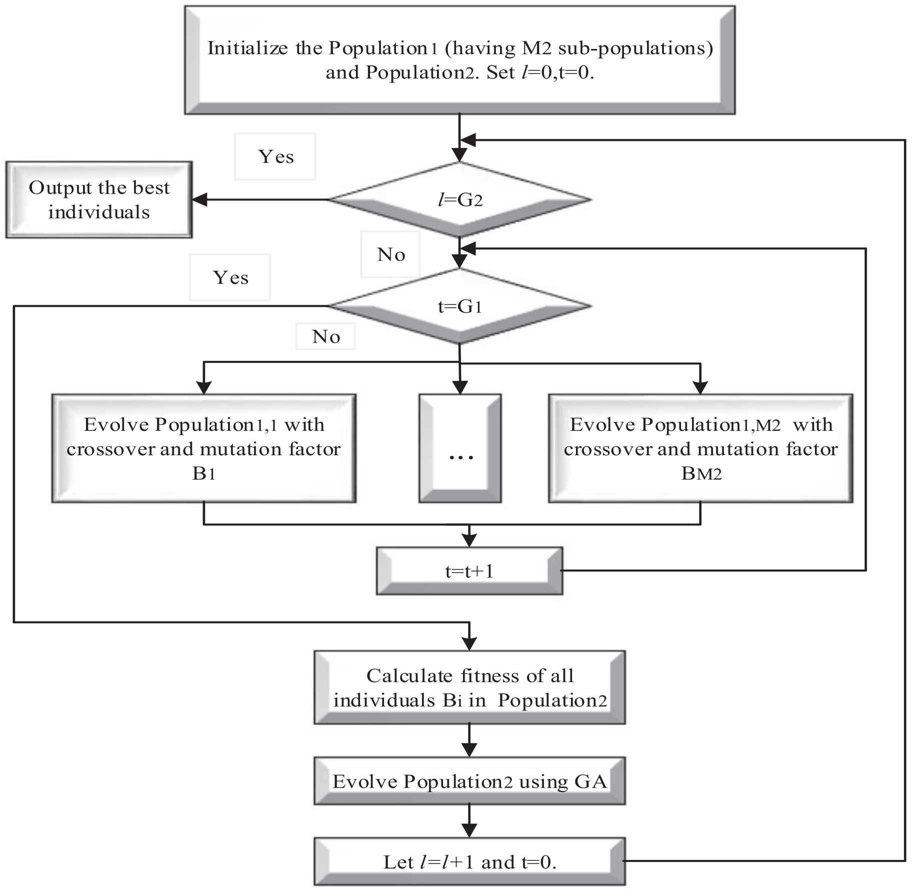

FIGURE 4 Flow chart of CGA^2 (see He and Wang[16]). GA indicates genetic algorithm

dependencies (ie, starting from the exit tasks). The HEFT algorithm generates schedules based on VMs and tasks and does not vary with constraints.

In our experiments, we model an IaaS provider offering a single data center and 5 types of VMs. The VM configurations are based on current Amazon EC2 offerings and are presented in Table 2. We set processing capacity of each type of VMs based on the work of Ostermann et al.[30]

The experiments are conducted by using 4 different deadlines. These deadlines lie between the slowest and fastest runtimes. The slowest runtime is obtained by using a single VM with the average processing capacity of all VMs to execute all tasks. And the fastest runtimes are obtained by assigning the highest processing capacity VM to the ready tasks. To estimate each of the 4 deadlines, we divide the difference between the fastest and slowest times by 10 to get an interval size. The first deadline is the slowest runtime minus 1 interval size to the fastest deadline, as to the second one, we minus 4 interval sizes. The third is the fastest runtime adding 2 interval sizes to the slowest deadline, and the last one is the fastest runtime adding 1 interval size.

For the testing, the parameters of CGA^2 are set as follows: $M_1 = 200$, $G_1 = 100$, $M_2 = 50$, and $G_2 = 20$. To compare the results, we consider the average workflow total execution cost and total execution time after running each experiment 30 times. All the experiments are performed on computers with Inter Core i5-4570S CPU (2.9 GHz and 8-GB RAM).

6.1 | Deadline Constraint Evaluation

In this section, we analyze the algorithms in terms of meeting the user's defined deadlines. We have compared the deadline meeting percentages for each scientific workflow under different deadlines as shown in Figure 6.

For the Epigenomics workflow, HEFT meets all of the deadlines. Random algorithm meets deadlines 1 and 2 at 10% and 3.3%, respectively, and completely fails to meet deadlines 3 and 4. The GA and PSO meet deadlines 1 and 2 at 100%, but when the constraints become strict, the rates become less and less. For deadline 3, the constraint meeting rates for the 2 evolutionary algorithms are 93% and 73.3%, respectively, and for deadline 4, the rates are 26.7% and 13.3%. As to our proposed CGA^2 algorithm, when the constraints become stricter, CGA^2 algorithm can still find excellent solutions in terms of constraints meeting. The deadline meeting rates for first 3 deadlines are 100%, and the deadline meeting rate is 80% for the last deadline constraint. The results for Montage application again show that HEFT meets all of the deadlines, and it is much better than that of other algorithms. In Montage, Random algorithm obtains results similar to those obtained in Epigenomics, and it meets all deadline constraints in the lowest rates. And the GA- and PSO-based algorithms perform well just when the deadline is relaxed like in deadlines 1 and 2. CGA^2 can still find excellent solutions in terms of constraints meeting when the constraints become stricter. The meeting rates in different constraints are 100%, 100%, 100%, and 60%.

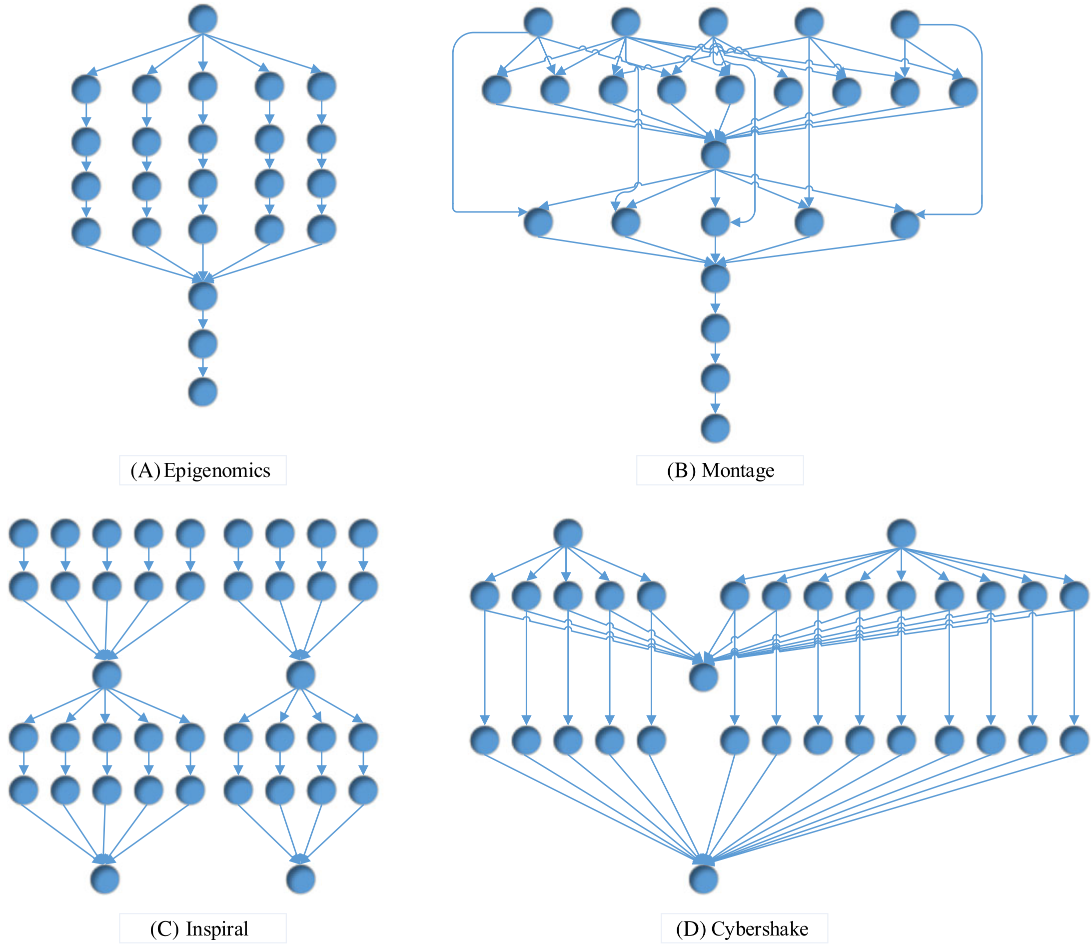

FIGURE 5 Structures of the 4 different workflows used in the experiments. A, Epigenomics. B, Montage. C, Inspiral. D, CyberShake

TABLE 2 Types of virtual machines used in the experiments

Name	EC2 Units	Processing Capacity	Cost per Hour ($)
m1.small	1	44	0.03
m1.large	4	176	0.12
m1.xlarge	8	352	0.24
c1.medium	5	220	0.06
c1.xlarge	20	880	0.44

The results of meeting rate for Inspiral and CyberShake are much like those of the above 2 workflows. The Random algorithm could hardly get feasible solutions under all constraints, while the HEFT gets 100% meeting rates under all constraints. As to the 3 evolutionary approaches, they perform similarly under the first 2 relaxed constraints, while our proposed algorithm obviously performs much better than the other 2 evolutionary algorithms. A possible explanation for these results revolves around the following: HEFT always assigns tasks with VMs that make the end time at a minimum level, and it considers the entire workflow rather than focusing on only unmapped independent tasks at each step to assign priorities to tasks. But it gives no concern to the cost and constraints. Random algorithm assigns tasks with VMs randomly, which is very hard to satisfy user's constraints. As to 2 evolutionary algorithms GA and PSO, they simply handle the constrained optimization problem, which just gives a static penalty function or even excludes the solutions that violate the constraints in the evolutionary process, and it would lead to a premature convergence or a false direction to the infeasible region.

6.2 | *TET* Evaluation

As shown in Figure 7, we measured the average of total execution time *TET* (also called as makespan) for each workflow under different constraints (red line is the *TET* constraint). We can see that in the Epigenomics workflow, the HEFT approach has the lowest average makespan under every deadline constraints, and the Random has the worst performance. The GA and the PSO have a good performance in the relaxed constraints like deadlines 1 and 2. And when the constraints become stricter, they perform poorly. As to our proposed CGA^2, it shows a better performance than both the GA and the PSO, especially in the strict constraints. In the Montage workflow case, the HEFT still has the best average makespan value while the Random is the worst in terms of total execution time. In deadlines 1 and 2, the mean of total execution time (*TET*) for the GA, PSO, and CGA^2 is less than deadlines, while the average *TET* for these algorithms in the last constraints is above the deadline values.

In the Inspiral and CyberShake workflows, our proposed algorithm CGA^2 still has better *TET* results than the other 2 evolutionary algorithms. These results are in line with those analyzed in the deadline-constraint evaluation section, from which we were able to conclude that HEFT algorithm can obtain the lowest total execution time value

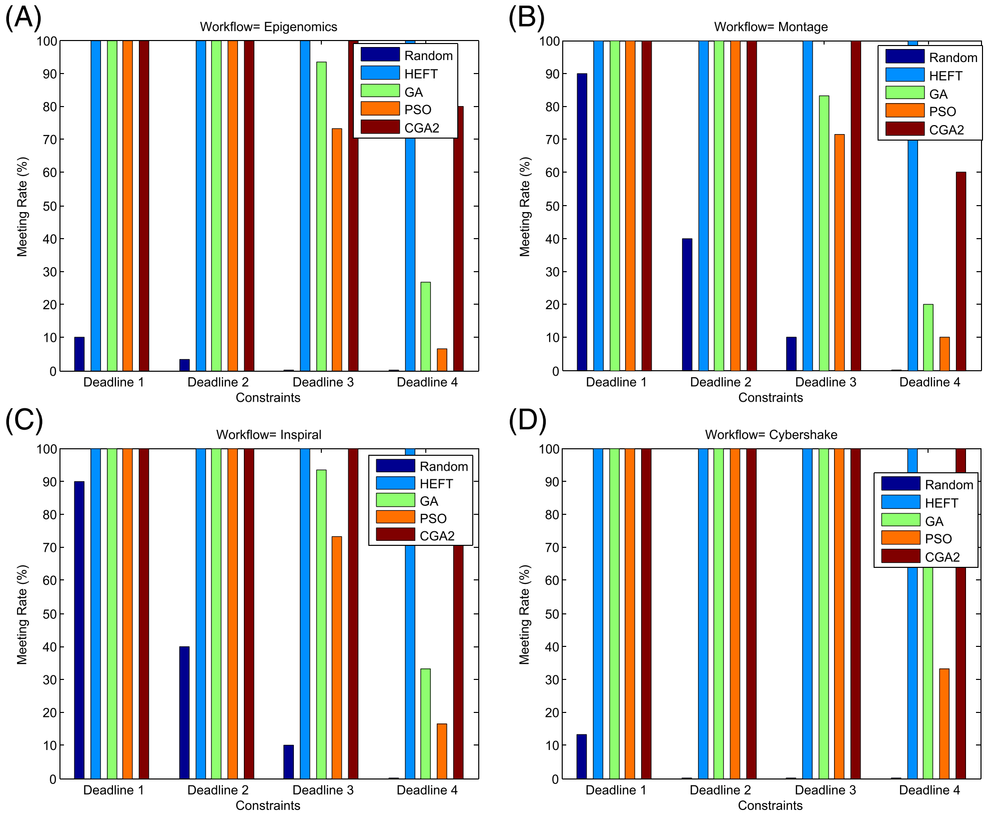

FIGURE 6 Deadlines meeting percentages for each scientific workflow. A, Epigenomics. B, Montage. C, Inspiral. D, CyberShake. GA indicates genetic algorithm; HEFT, Heterogeneous Earliest Finish Time; PSO, particle swarm optimization

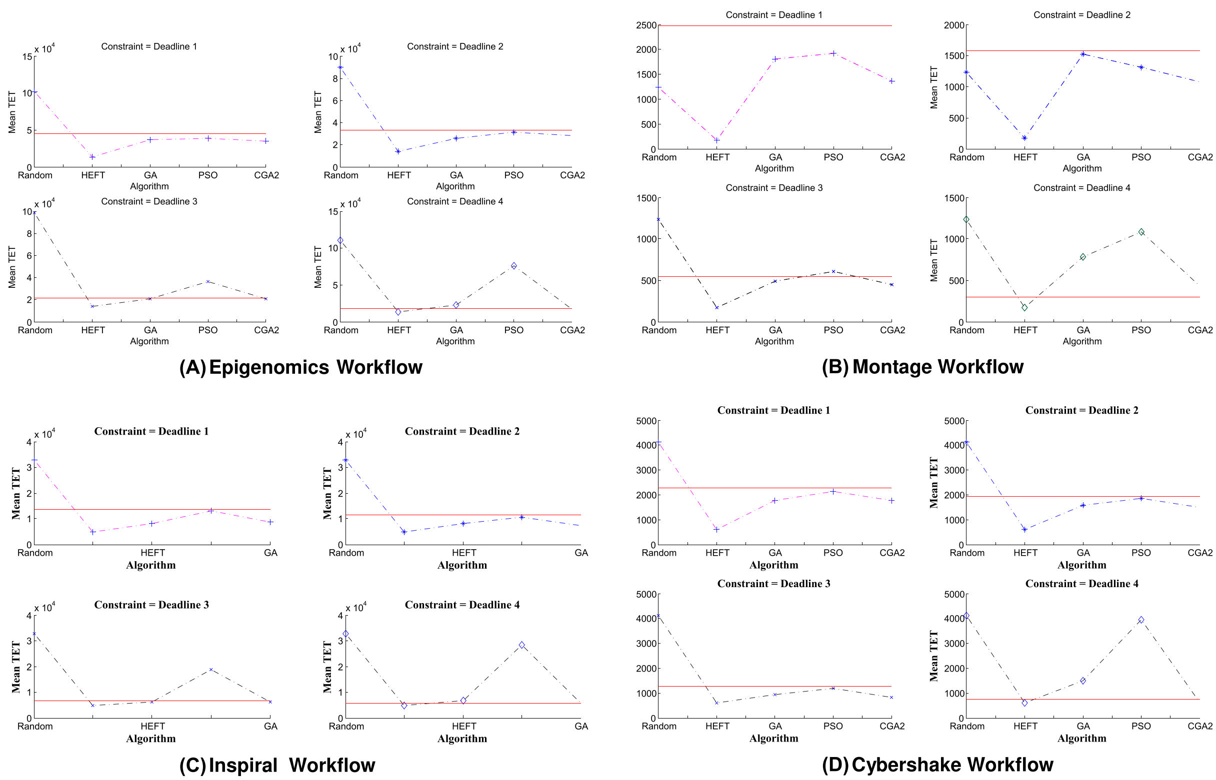

FIGURE 7 Average of *TET* (dash line) under different constraints (solid line) for 4 workflows. A, Epigenomics. B, Montage. C, Inspiral. D, CyberShake. GA indicates genetic algorithm; HEFT, Heterogeneous Earliest Finish Time; PSO, particle swarm optimization

for workflows, that Random algorithm is not efficient in meeting the deadlines, and that the normal GA and PSO cannot find excellent solutions when the constraints become stricter. On the contrary, CGA^2 exhibits a larger makespan variation, which is expected as it can find an excellent solution for constrained optimization problem in a very large and chaotic search space.

6.3 | *TEC* Evaluation

The average total execution costs generated from the above algorithms for each of the workflows are displayed in Figure 8. We also give the mean of *TET* and deadlines meeting rates obtained in the former section, as the algorithms should be able to generate a cost-efficient scheme but not at the expense of a long execution time. It is unavailable for an algorithm to run on the lower cost without meeting the deadlines. The mean of *TEC*, the mean of *TET*, and the meeting rate for each workflow under different constraints are presented in Table 3.

For the Epigenomics workflow, HEFT total execution time is still the lowest of the 4 different constraints, but its total execution cost is higher than that of evolutionary algorithms. In these experiments, CGA^2 gets the lowest cost under each deadline, which shows that the optimizing capacity of our proposed algorithm is better than that of other previous evolutionary algorithms especially under the strict deadlines. From the table, we can also find that when the constraints become stricter, the solutions obtained by the GA and PSO not only fail to meet deadlines but also become much more costly. It shows that an inferior penalty function would lead the populations to premature and infeasible sections.

In the Montage workflow, HEFT total execution time is still the lowest for the 4 different constraints, but its total execution cost is higher than that of evolutionary algorithms. A possible explanation for this might be that evolutionary algorithms lease VMs with lower price so as to minimize the total execution cost. What is more, the tasks are relatively small in Montage workflow, which means that the machines in the HEFT scheme are only running for a small amount of time but are charged for the full billing period, and choosing a higher processing capacity VM means much more cost.

Among the above algorithms complying with the deadline constraint, GA and PSO can obtain the low cost schedules on relaxed deadline, but when the constraints become strict, the solutions generated by them are very poor, while the CGA^2 can still get excellent solutions in terms of the *TEC* meeting deadlines under strict constraints. From the results, it is clear that the evolutionary algorithms based approaches perform better than HEFT in terms of cost. And the CGA^2 can get an excellent solution without violating the constraints when the deadline becomes strict.

The *TEC* results of 5 algorithms in the Inspiral and CyberShake workflows are similar to those of the previous workflows, which again show that CGA^2 could always find the best solutions in terms of *TEC*, especially under the strict constraints. We can observe from Figure 8C and D that the GA and PSO with static penalty

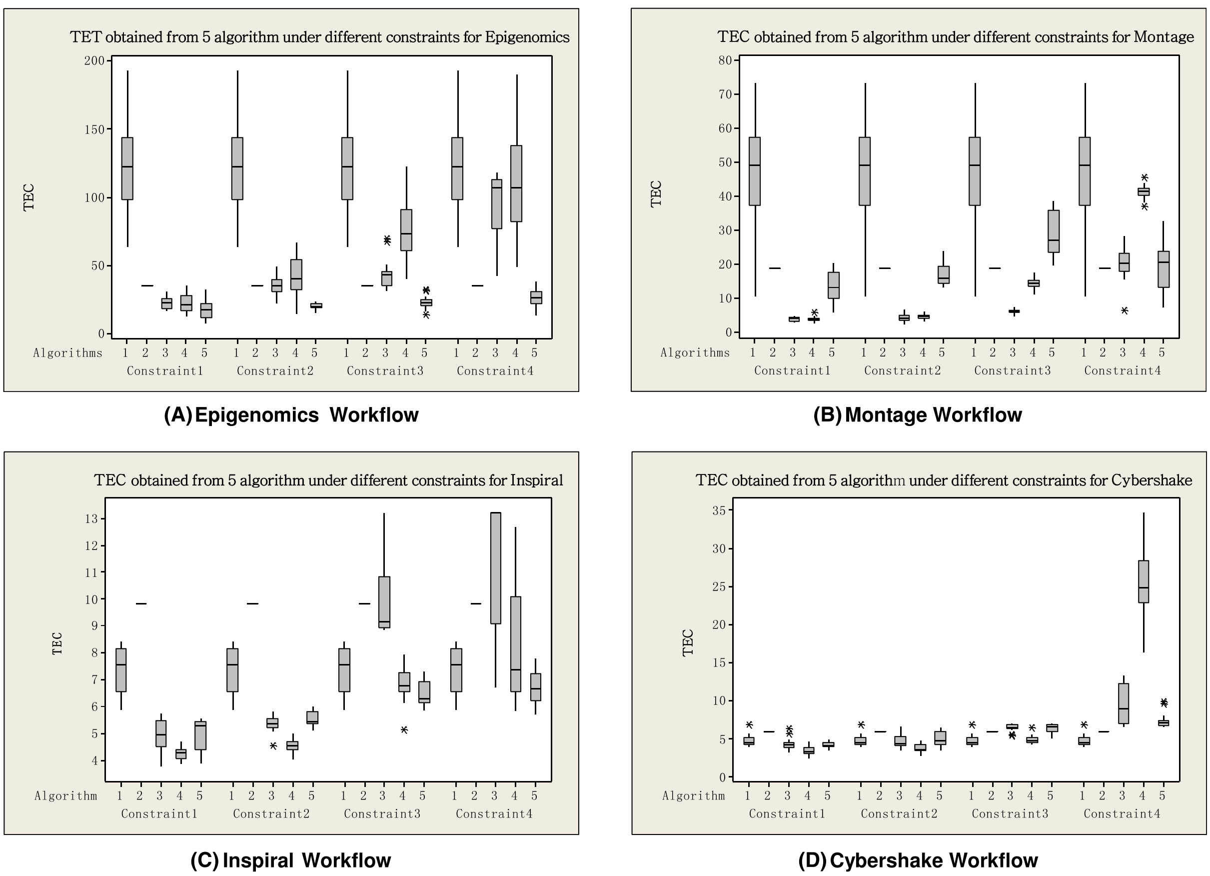

FIGURE 8 *TEC* generated by above algorithms under different constraints for 4 workflows. A, Epigenomics. B, Montage. C, Inspiral. D, CyberShake

TABLE 3 Performance comparison of algorithms under different constraints

Workflow	Algorithm	Mean *TEC*	95% CI *TEC*	Mean *TET*	Meeting Rate (%)	Mean *TEC*	95% CI *TEC*	Mean *TET*	Meeting Rate (%)
			Deadline 1				Deadline 2		
Epigenomics	Random	100.35	[40.2, 160.5]	98 475	10	100.35	[40.2, 160.5]	102 099	3.3
	HEFT	35.12	[35.12, 35.1]	13 916	100	35.10	[35.12, 35.12]	13 916	100
	GA	22.73	[12.9, 32.5]	20 743	100	32.98	[12.5, 54.5]	25 892	100
	PSO	21.15	[9.2, 33.1]	36 140	100	41.55	[8.1, 85.1]	31 170	100
	CGA^2	17.95	[7.2, 28.7]	20 919	100	20.38	[10.7, 20.1]	28 404	100
Montage	Random	52.56	[24.4, 80.7]	1233.3	90	52.56	[24.38, 80.74]	1233.3	40
	HEFT	18.65	[18.7, 18.7]	173.45	100	18.65	[18.65, 18.65]	173.45	100
	GA	4.03	[2.45, 5.11]	1806.2	100	3.98	[2.33, 5.63]	1521.0	100
	PSO	13.7	[6.3, 21.1]	1922	100	16.21	[10.4, 22.0]	1315.6	100
	CGA^2	3.64	[2.43, 4.85]	1362	100	4.82	[3.05, 6.60]	1082	100
Inspiral	Random	52.56	[6.93, 7.82]	32 762.0	90	52.56	[24.38, 80.74]	32 762	40
	HEFT	9.83	[9.83, 9.83]	4783.56	100	9.83	[9.83, 9.83]	4783.5	100
	GA	4.96	[4.64, 5.27]	8058.10	100	5.33	[5.16, 5.50]	8189.0	100
	PSO	4.24	[4.13, 4.35]	12 900.0	100	4.52	[4.40, 4.65]	10 610	100
	CGA^2	4.94	[4.63, 5.25]	8678.10	100	5.54	[5.38, 5.69]	7129.7	100
CyberShake	Random	9.8000	[9.38, 10.22]	4130.90	13.3	9.8000	[9.38, 10.22]	4130.90	0
	HEFT	8.86	[8.86, 8.86]	601	100	8.86	[8.86, 8.86]	601.00	100
	GA	4.31	[3.85, 4.78]	1776.80	100	4.6980	[4.11, 5.28]	1567.40	100
	PSO	3.41	[3.04, 3.77]	2133.00	100	3.7373	[3.45, 4.02]	1858.00	100
	CGA^2	4.15	[3.91, 4.39]	1767.50	100	4.9487	[4.43, 5.47]	1483.70	100
			Deadline 3				Deadline 4		
Epigenomics	Random	98.37	[40.2, 160.5]	98 475	0	87.43	[40.2, 160.5]	110 509	0
	HEFT	35.12	[35.12, 35.1]	13 916	100	35.12	[35.12, 35.12]	13 916	100
	GA	46.98	[20.8, 73.6]	20 743	93.3	97.69	[54.2, 141.3]	22 745	73.3
	PSO	80.28	[28.5, 132.1]	36 140	26.7	124.16	[65.9, 184.4]	76 206	6.7
	CGA^2	23.06	[15.2, 30.9]	20 919	100	25.28	[11.7, 38.8]	16 807	80
Montage	Random	52.56	[24.38, 80.7]	1233.3	10	52.56	[24.38, 80.74]	1233.3	0
	HEFT	18.65	[18.65, 18.7]	173.45	100	18.65	[18.65, 18.65]	173.45	100
	GA	13.6	[8.64, 18.62]	486.4	80	21.65	[11.63, 31.7]	779.5	20
	PSO	28.9	[16.8, 40.1]	611.6	70	41.6	[35.7, 46.5]	1085.5	10
	CGA^2	5.87	[4.05, 7.69]	448.9	100	18.9	[8.9, 30.9]	435	60
Inspiral	Random	52.56	[24.4, 80.7]	32 762.0	10	52.56	[24.38, 80.74]	32 762	0
	HEFT	9.83	[9.83, 9.83]	4783.5	100	9.83	[9.83, 9.83]	4783	100
	GA	9.99	[9.25, 10.73]	6233.3	93.3	11.457	[10.0, 12.9]	6803.7	33.3
	PSO	6.77	[6.42, 7.13]	18 650.0	73.3	8.1353	[7.02, 9.24]	28 300	16.6
	CGA^2	6.46	[6.21, 6.71]	6247.6	100	6.6580	[6.31, 7.00]	5504.5	96.7
CyberShake	Random	9.8000	[9.38, 10.22]	4130.90	0	9.8000	[9.38, 10.22]	4130	0
	HEFT	8.86	[8.86, 8.86]	601.00	100	8.86	[8.86, 8.86]	601.00	100
	GA	6.4400	[6.19, 6.68]	922.96	100	10.796	[8.82, 12.77]	922.96	86.7
	PSO	4.8933	[4.56, 5.22]	1200.00	100	25.001	[22.5, 27.47]	1200.0	33.3
	CGA^2	6.3680	[6.04, 6.69]	815.99	100	7.4060	[6.84, 7.96]	815.99	100

Abbreviations: GA, genetic algorithm; HEFT, Heterogeneous Earliest Finish Time; PSO, particle swarm optimization.

function have a rapid increase in terms of *TEC* when the constraints get stricter, while our algorithm is able to maintain a similar level. What is more, even though the PSO with static penalty function could get better *TEC* results than CGA^2 in the relaxed constraints, it still could not get satisfactory solutions in the strict constraints.

From the above discussion, the following conclusions can be drawn from the experiments: the Random algorithm fails to meet deadlines in most cases, whereas HEFT could get the lowest makespan, which always assigns the highest processing capacity VMs to ready tasks. While comparing with HEFT, the evolutionary algorithms are capable of generating cheaper schedules and hence outperform HEFT in terms of cost optimization. And compared with GA and PSO with static penalty functions, our proposed algorithm CGA^2 can handle better the constrained optimization problem of scientific workflow scheduling in clouds and is still able to find feasible solutions under strict user's deadline requirements.

7 | CONCLUSION AND FUTURE WORK

In this paper, a CGA with adaptive penalty function (CGA^2) for constrained scientific workflow scheduling in clouds is proposed. The common drawback of existing evolutionary algorithms is the necessity of defining problem-specific parameters of penalty function for constrained optimization problem. And these algorithms are also static and lead to premature convergence. To address these problems, our proposed algorithm designs an adaptive penalty function without any parameter tuning and is easy to implement.

This approach could effectively exploit the information hidden in infeasible individuals, which sets the proper penalty rule for the individuals at different stages of the evolutionary process. In addition, our proposed algorithm CGA^2 applies adaptive crossover and mutation probabilities based on the coevolution and generates the initial population according to CP, which are efficient for preventing premature and improving deadline meeting for workflow scheduling. Experiments demonstrate that our solution has an overall better performance than the state-of-the-art algorithms Random, HEFT, GA, and PSO. The CGA^2 succeeds with high rate as HEFT, which aims to minimize the makespan without considering the cost. Furthermore, CGA^2 could produce schedules with lower total executing cost and meets the deadlines under strict constraints, while GA and PSO could not succeed easily.

Our future work will use multiobjective evolutionary algorithm to solve the cloud resource scheduling problem and will take into account the load balance and task failures. Meanwhile, we will extend the resource model to consider the data transfer cost between data centers.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (grant nos. 61370132, 61472033, and 61272432) and Beijing Natural Science Foundation (no. 4152034).

REFERENCES

1. Vöckler JS, Juve G, Deelman E, Rynge M, Berriman B. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ACM, 2011; 15–24. DOI:10.1145/1996109.1996114.
2. http://lhc.web.cern.ch/lhc/LHCExperiments.htm. Accessed January 8, 2016.
3. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst*. 2009;25(6):599–616. doi: 10.1016/j.future.2008.12.001.
4. Foster I, Zhao Y, Raicu I, Lu S. Cloud computing and grid computing. 360-degree compared. In *Grid Computing Environments Workshop, GCE'08*, IEEE, 2008; 1–10. DOI: 10.1109/GCE.2008.4738445.
5. Liu L, Zhang M, Lin Y, Qin L. A survey on workflow management and scheduling in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 14th IEEE/ACM International Symposium on*, IEEE, 2014; 837–846. DOI: 10.1109/CCGrid.2014.83.
6. Rodriguez MA, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans Cloud Computing*. 2014;2(2):222–35. doi: 10.1109/TCC.2014.2314655.
7. Ma T, Buyya R. Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids. In *Computer Architecture and High Performance Computing, SBAC-PAD 2005. 17th International Symposium on*, IEEE, 2005; 251–258. DOI: 10.1109/CAHPC.2005.22.
8. Wang X, Yeo CS, Buyya R, J S. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. *Future Gener Comput Syst*. 2011;27(8):1124–34. doi: 10.1016/j.future.2011.03.008.
9. Pandey S, Wu L, Guru SM, Buyya R. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced information networking and applications (AINA), 24th IEEE international conference on*, IEEE, 2010; 400–407. DOI: 10.1109/AINA.2010.31.
10. Sawant S. A genetic algorithm scheduling approach for virtual machine resources in a cloud computing environment, 2011.
11. Huang J. The workflow task scheduling algorithm based on the GA model in the cloud computing environment. *J Softw*. 2014;9(4):873–80.
12. Feller E, Rilling L, Morin C. Energy-aware ant colony based workload placement in clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, IEEE Computer Society, 26–33. DOI: 10.1109/Grid.2011.13.
13. Wu Z, Ni Z, Gu L, Liu X. A revised discrete particle swarm optimization for cloud workflow scheduling. In *Computational Intelligence and Security (CIS), International Conference on*, IEEE, 2010; 184–188. DOI: 10.1109/CIS.2010.46.
14. Paredis J. Co-evolutionary constraint satisfaction. In: *Parallel Problem Solving from Nature—PPSN III*. Berlin Heidelberg: Springer; 1994:46–55.
15. Coello CAC. Use of a self-adaptive penalty approach for engineering optimization problems. *Comp Industry*. 2000;41(2):113–27. doi: 10.1016/S0166-3615(99)00046-9.
16. He Q, Wang L. An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Eng Appl Artif Intel*. 2007;20(1):89–99. doi: 10.1016/j.engappai.2006.03.003.
17. Goldberg DE. *Genetic Algorithms in Search Optimization and Machine Learning*. Vol. 412. Reading Menlo Park: Addison-wesley; 1989.
18. Zhang J, Chung HSH, Lo WL. Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Trans Evolut Comput*. 2007;11(3):326–35. doi: 10.1109/TEVC.2006.880727.
19. Li YL, Shao W, Wang JT, *et al.* An improved NSGA-II and its application for reconfigurable pixel antenna design. *Radio Eng*. 2014;23(2):733–8.
20. Srinivas M, Patnaik LM. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Trans Syst Man Cybern*. 1994;24(4):656–67.
21. Tessema B, Yen GG. A self adaptive penalty function based algorithm for constrained optimization. In *Evolutionary Computation, CEC 2006. IEEE Congress on*, IEEE, 2006; 246–253. DOI: 10.1109/CEC.2006.1688315.
22. Coello CA. *Use of a self-adaptive penalty approach for engineering optimization problems*. Computers in Industry; 2000:113–27.
23. Nanakor P, Meesomklin K. An adaptive penalty function in genetic algorithms for structural design optimization. *Comput Struct*. 2001;79(29):2527–39. doi: 10.1016/S0045-7949(01)00137-7.
24. Tessema B, Yen GG. An adaptive penalty formulation for constrained evolutionary optimization. *IEEE Trans Syst Man Cybern A, Syst Humans*. 2009;39(3):565–78. doi: 10.1109/TSMCA.2009.2013333.
25. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Pract Ex*. 2011;41(1):23–50. doi: 10.1002/spe.995.
26. Chen W, Deelman E. WorkflowSim: a toolkit for simulating scientific workflows in distributed environments. In *E-Science (e-Science), IEEE 8th International Conference on*, IEEE, 2012; 1–8. DOI: 10.1109/eScience.2012.6404430.
27. Zhu Z, Zhang G, Li M, *et al.* Evolutionary Multi-Objective Workflow Scheduling in Cloud. *IEEE Trans Parallel Distr Syst*. 2016;27(5):1344–57.
28. Rahman M, Hassan R, Ranjan R, Buyya R. Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency Comput Pract Ex*. 2013;25(13):1816–42. doi: 10.1002/cpe.3003.
29. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distr Syst*. 2002;13(3):260–74. doi: 10.1109/71.993206.
30. Ostermann S, Iosup A, Yigitbasi N, Prodan R, Fahringer T, Epema D. A performance analysis of EC2 cloud computing services for scientific computing. In: *Cloud computing*. Berlin Heidelberg: Springer; 2009: 115–31.

How to cite this article: Liu, L., Zhang, M., Buyya, R., and Fan, Q. (2016), Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing, *Concurrency Computat.: Pract. Exper.*, doi: 10.1002/cpe.3942