J Grid Computing (2014) 12:637–663
DOI 10.1007/s10723-014-9306-7

Efficient Workflow Scheduling for Grid Computing Using a Leveled Multi-objective Genetic Algorithm

Hassan Khajemohammadi · Ali Fanian · T. Aaron Gulliver

Received: 22 July 2013 / Accepted: 19 May 2014 / Published online: 21 August 2014

Abstract Task scheduling and resource allocation are important problems in grid computing. The workflow management system receives inter-dependent tasks from the users and allocates each task to an appropriate resource based on user requirements and constraints such as budget and deadline. Thus this system has a significant effect on performance and the efficient use of resources. In general, mapping tasks to distributed resources is an NP-hard problem. Hence, heuristic and meta-heuristic methods are typically employed. Moreover, since tasks can enter the system at any time, the task scheduling runtime is an important parameter for workflow management systems. This paper presents a fast method for scheduling workflows in a grid environment based on a multi-objective Genetic Algorithm (GA). In the proposed method, the workflows and chromosomes in the GA are assigned to levels to reduce the scheduling time. In addition, the proposed method prevents infeasible solutions being produced in new generations, so task dependencies do not need to be checked. New crossover and mutation operators are proposed to improve convergence and maintain solution diversity. Experimental results are presented and evaluated using several well-known metrics as well as a new metric. This shows the effectiveness of the proposed method compared to other approaches.

Keywords Workflow scheduling · Genetic algorithm · Multi-objective optimization · Grid computing

H. Khajemohammadi · A. Fanian (✉)
Department of Electrical and Computer Engineering, Isfahan University of Technology (IUT), Isfahan, Iran
e-mail: a.fanian@cc.iut.ac.ir

H. Khajemohammadi
e-mail: h.khajemohammadi@ec.iut.ac.ir

T. A. Gulliver
Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada
e-mail: agullive@ece.uvic.ca

1 Introduction

Computational grids are an alternative to supercomputers for solving large scale problems [1]. They can be divided into two groups: utility grids and community grids [2, 3]. The resources in utility grids have characteristics such as Quality of Service (QoS) and cost which allow users to consume resources appropriately. In a community grid, resources are available according to a best effort model and cost is not a consideration. Utility grids allow users to reserve resources to ensure the availability of services, and the QoS can be negotiated [4]. Conversely, the QoS and service availability in community grids may not be guaranteed [5]. In recent years, utility computing services have improved due to the development of service-oriented grid computing. This provides a

transparent infrastructure that enables users to obtain secure, scalable, and standard services in a global network environment [6, 7].

An important issue in the grid computing environment is workflow scheduling. In the literature, several methods have been proposed to address the job scheduling problem in community grids. However, very few techniques have been developed for scheduling in utility grids. Most grid applications require the execution of a large number of interdependent jobs in areas such as biotechnology, astronomy, and weather forecasting. Workflow Management Systems (WfMSs) such as Pegasus [3], ASKALON [8], GrADS [9] and others [10–14], have been developed to define, manage and execute workflows in grid environments. Towards this end, the WfMS taxonomy has been proposed [15–17].

The main issues in workflow scheduling are how to select an appropriate resource for each job and determine the execution order of jobs on resources to meet user requirements such as QoS. Existing workflow management systems attempt to minimize the workflow execution time. However, in utility grids the cost of executing jobs on various services should also be considered. Typically, services with a higher QoS cost more. The key issues in workflow scheduling are as follows [18]:

- Resources are usually shared between users and there may be competition between them for these resources.
- The scheduler is not in control of the resources.
- The available resources vary over time.
- Resources are located on multiple management sites.
- Resources are heterogeneous.
- Most workflow applications are data-centric and therefore a large amount of data may be transferred between sites.

As mentioned above, satisfying user requirements such as the cost and delay of workflow execution is an important problem in grid computing. In this paper, a new workflow scheduling algorithm is proposed which considers user requirements. Each workflow is first associated with a level according to the dependency between jobs, and then a multi-objective genetic algorithm is used to perform the scheduling. The main goal of this algorithm is not only to optimize workflow execution time and cost but also minimize the scheduling time which has a significant impact on grid performance.

The remainder of the paper is organized as follows. Section 2 presents related work on the workflow scheduling problem and multi-objective workflow scheduling. The workflow scheduling model and multi-objective optimization are also presented. The proposed scheduling algorithm is described in Section 3. Section 4 presents some performance results to illustrate the advantages of this algorithm, and finally some conclusions are given in Section 5.

2 Preliminaries

2.1 Related Work

Workflow scheduling focuses on the mapping and execution management of interdependent tasks on services. In general, the problem of mapping tasks to distributed resources is an NP-hard problem [5, 19], so algorithms that can find an optimal solution in polynomial time are not known. Hence, heuristic and meta-heuristic methods are used to find appropriate solutions in homogeneous [6] and heterogeneous [20, 21] distributed systems such as grids. For example, heuristics based on nature have been used for task scheduling [22–24]. A Genetic Algorithm (GA) and the Heterogeneous Earliest Finish Time (HEFT) algorithm were employed in the ASKALON project for job scheduling [8]. Most of the proposed techniques for workflow scheduling consider only one objective, and very few attempt to optimize several objectives simultaneously.

To address the scheduling problem in grids which are heterogeneous distributed systems, several objectives are considered such as makespan, reliability of the scheduling, required budget to execute jobs, utilization of the resources, and fault tolerance. Some of these objectives have high priority from the user point of view while others are important to resource providers. Therefore, some methods have been proposed based on a single objective while others consider a combination of objectives. In [25], the authors enhanced an existing list scheduling algorithm designed to minimize the workflow

makespan with advance reservation-based negotiation functionality. They illustrate through real-world experiments two benefits of their approach: improved execution predictability from the user perspective and higher resource utilization fairness through a new progressive allocation strategy from the provider perspective. The multi-criteria approach proposed in [26] considers job completion time and security risk as the main objectives of the independent job scheduling process. They changed the implementation of a traditional genetic algorithm and proposed the Accelerated Genetic Algorithm (AGA). This provides online job scheduling with quick convergence to optimal solutions.

The method presented in [27] proposes a dependable grid workflow scheduling system. It employs a Markov chain-based resource availability prediction model. Based on this model, a reliability cost driven workflow scheduling algorithm was developed. It first predicts the reliability of a resource node during task execution and then makes a scheduling decision in terms of the reliability cost of successfully executing the task. A distributed scientific workflow mapping algorithm for maximizing reliability under an End-to-End Delay (EED) bound was proposed in [28]. This algorithm considers both the maximum reliability and the minimum EED objectives in a two-step procedure. In the first step, a mapping algorithm combining iterative critical path search and layer-based priority assignment techniques is adopted to minimize the EED by focusing on the optimal allocation of tasks on the critical path. In the second step, tasks on noncritical paths are remapped to improve the overall execution reliability.

In [29], a GA was used to optimize several objectives concurrently. This method simply searches points in the solution space (called reference points), according to user demands rather than conducting a thorough search. Thus solutions are found quickly, but the search space is quite limited. The method proposed in [30] uses different heuristics to generate an initial population. As a result, good solutions are obtained using less iteration than with a random population, but the complexity is higher. In [31], workflow scheduling based on a Multi-objective Differential Evolution (MODE) algorithm was proposed to obtain a diverse distribution of solutions in the solution space. To achieve this goal, three parents are employed to create offspring, and the Ulam distance [32] is used to calculate the quality of these parents. A scheduling algorithm based on a multi-objective GA was presented in [33] and its performance compared with that of the Min-Min and Min-Max [33] algorithms. In [34], a trustworthiness method based on an Integer Genetic Algorithm (IGA) was proposed for scheduling business applications. Reliability was considered as a user QoS parameter in addition to execution time and cost.

2.2 The Workflow Scheduling Model

A workflow is modeled as a Directed Acyclic Graph (DAG) to schedule N interdependent tasks. In this model, let Γ be a limited set of tasks T_{i} $(1 \leq i \leq N)$, and Λ be a collection of directed edges $e_{i,j} = (T_{\mathrm{i}}, T_{\mathrm{j}})$ where T_{i} is the parent of T_{j} and T_{j} is the child of T_{i}. A task without a parent is called an entry task, and a task without a child is called an exit task. If a workflow has more than one entry task or exit task, new dummy tasks T_{entry} and T_{exit} are added to the beginning and end of the workflow as required. These tasks have no jobs to execute and no data to transfer. It is assumed that a child cannot be executed until all of its parents have finished.

Let β and $\mathcal{D}$ denote the workflow budget and deadline, respectively, specified by a user. Then, the workflow can be expressed as $\Omega\ (\Gamma, \Lambda, \beta, \mathcal{D})$. Let m be the number of available services. For each task T_{i} there is a set of services that are able to execute it, but only one service $S_i^j = (1 \leq i \leq N, 1 \leq j \leq m_i, m_i \leq m)$, can be selected to execute a task. Let $\mathrm{ET}_{\mathrm{I}}\left(T_i, S_i^j\right)$ and $\mathrm{EC}_{\mathrm{I}}\left(T_i, S_i^j\right)$ be the completion time and cost for task T_{i} on service S_i^j in solution I, respectively. Estimating the execution time of each task on a desired service is important for scheduling. Several methods such as code analysis, analytical benchmarking/code profiling, and statistical prediction have been proposed for this purpose [35]. Data transfer also consumes time and has a cost. Let $\mathrm{TT}_{\mathrm{I}}(e_{i,j}, r, s)$ and $\mathrm{TC}_{\mathrm{I}}(e_{i,j}, r, s)$ be the data transfer time and cost for solution I, respectively, for data to be transferred from service r (task T_{i}) to service s (execute task T_{j}) via edge $e_{i,j}$. This time can be estimated based on the size of the data and the available bandwidth between services.

The purpose of scheduling is to assign each task T_{i} to a suitable service $S_{\mathrm{i}}^{\mathrm{j}}$ with the aim of minimizing the

workflow execution cost and time while satisfying the user requirements. The optimization problem is then

$$Minimize\, F(I) = (f_1(I), f_2(I)) \tag{1}$$

$$f_1(I) : Min\, Time(I) = \underset{T_i \in \Gamma}{\text{Max}} \left(ET_1\left(T_i, S_i^j\right) + \sum_{e_{i,j} \in \Lambda} TT_I\left(e_{i,j}, r, s\right) \right) \tag{2}$$

$$f_2(I) : Min\, Cost(I) = \sum_{T_i \in \Gamma} EC_I\left(T_i, S_i^j\right) + \sum_{e_{i,j} \in \Lambda} TT_I\left(e_{j,j}, r, s\right)$$

$$subject\ to : f_1(I) < \mathcal{D}, f_2(I) < \beta \tag{3}$$

where $I \in X$, X is the solution space, f_1 is the workflow execution time function, f_2 is the workflow execution cost function, $\mathcal{D}$ is the user time constraint (deadline), β is the user cost constraint (budget), and $f_i(I)$ is the value of objective function i for solution I.

2.3 Multi-objective Optimization

A Genetic Algorithm (GA) is an optimization technique for solving complex problems [36]. Further, a multi-objective GA considers several objectives, which may be conflicting, to obtain a problem solution [37]. From (1), (2), and (3), workflow scheduling has two conflicting objectives, minimizing execution time and minimizing execution cost, and there may be several possible solutions [38]. In general, a multi-objective optimization problem can be expressed as [31].

$$\text{Minimize } F(I) = f_1, \ldots, f_n(I) \tag{4}$$

Among the solutions obtained, one solution I^* dominates the others if it is as good as or better than the other solutions for all objective functions [31], i.e.

$$\forall i \in [1, \ldots, n];\ f_i(I^*) \leq f_i(I) \bigwedge \exists j \in [1, \ldots, n],\ f_i(I^*) < f_i(I) \tag{5}$$

A solution is said to be *Pareto optimal* if it is not dominated by any other solution in the solution space. A Pareto optimal solution cannot be improved with respect to any objective without worsening at least one other objective. The set of all feasible non-dominated solutions in a solution space is referred to as the *Pareto optimal set*, and for a given Pareto optimal set the corresponding objective function values in the objective space are called the *Pareto front*. For many problems, the number of Pareto optimal solutions is enormous (perhaps infinite). The ultimate goal of a multi-objective optimization algorithm is to identify solutions in the Pareto optimal set [38].

A genetic algorithm is a reliable search technique based on the principles of evolution which can find good solutions in a large solution space in polynomial time [39]. Each solution in the search space is an individual (chromosome), and previous solutions are combined to generate new solutions. A fitness function is used to determine the quality of the new solutions. In general, a genetic algorithm is composed of the following steps [39]:

1. Randomly generate an initial population of size N.
2. Create new offspring (new solutions) using genetic operators on selected parents (previous solutions).
3. Evaluate the fitness of each new solution.
4. Select N solutions for the next generation.
5. Repeat steps 2 through 4 until the stopping condition is satisfied.

3 The Proposed Algorithm

As discussed above, several heuristic algorithms have been proposed to optimize workflow execution time or cost. Several multi-objective GA based methods have been developed such as PAES [40], NSGA-II [41], VEGA [42], and SPEA2 [43]. They differ in terms of the features and components employed such as diversity, elitism and population [38]. In addition to the user constraints, it is important that the solutions generated satisfy the job dependencies, i.e., each child in the DAG can be executed only if all of its parents have finished. Due to the large number of chromosomes (solutions) generated each generation, checking these dependencies for each new population is time consuming, and so will decrease workflow system performance. Therefore, a goal of the proposed

algorithm is to limit the time required to make scheduling decisions, including dependency checking.

Since a workflow can consist of a number of interrelated jobs, the GA for workflow scheduling should prevent the generation of infeasible solutions and also ensure the dependencies are satisfied. Thus a new scheduling algorithm called Level Workflow Scheduling based on a Genetic Algorithm (LWSGA) is proposed to reduce the workflow scheduling time while satisfying user constraints. In this algorithm, a DAG is considered as a collection of levels. Then a workflow is a collection of interdependent jobs with each job belonging to a level. Figure 1 shows a workflow divided into 5 levels. Tasks on a level are scheduled only after the previous level has completed to ensure all children are scheduled after their parents. This eliminates the need to check task dependencies and so will reduce workflow scheduling time. A modified multi-objective genetic algorithm is presented in Algorithm 1 to implement the LWSGA algorithm. This algorithm is discussed in detail in the remainder of this section.

3.1 Problem Formulation

Scheduling problems must be in a form suitable for Algorithm 1. In the proposed structure, each chromosome represents one scheduling solution consisting of a sequence of tasks and their associated services. Solutions are feasible if they satisfy the following conditions:

1- Each task can be executed only after all of its parent tasks and all tasks in previous levels have completed.

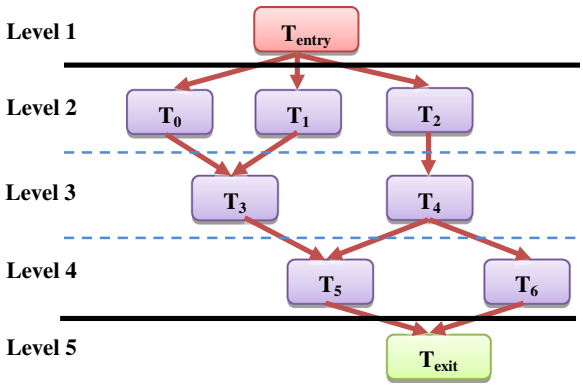


Fig. 1 An example of a workflow divided into levels

2- Each task appears only once in a schedule.
3- Each task is allocated to one time slot in a service.
4- Each task is assigned to an appropriate workflow level.

After the workflow is divided into levels, the chromosome structure is determined based on these levels. Each chromosome has Execution Order (EO) and Service Mapping (SM) fields. The EO specifies the execution order of each task in each level, and the SM specifies the services that will execute the tasks. For each task in a workflow, one time slot of the selected service will be reserved to execute the corresponding task. Figure 2 shows the chromosome structure for the workflow in Fig. 1. Both T_{entry} and T_{exit} are dummy tasks, so no services are assigned to them. If two tasks want access to the same service at the same time, the task that comes earlier in the EO has priority. For example, in the workflow shown in Fig. 1, tasks T0 and T4 may be executed concurrently, and according to Fig. 2, service S2 is assigned to both T0 and T4. In this case T0 will use S2 first.

3.2 Initial Population

In a genetic algorithm, the initial population can have a significant effect on performance. If the initial population contains infeasible solutions, it can consume an excessive amount of time and thus limit the number of solutions obtained. In the proposed method, the initial population is generated based on workflow levels, task requirements, and resource availability to prevent infeasible solutions. The algorithm for creating the initial population is shown in Algorithm 2. In Algorithm 2, the workflow levels and chromosome structure are considered so as to prevent infeasible solutions. The output is an initial population with N chromosomes.

3.3 Fitness Function

The fitness function is used to measure the quality of the solutions. Here the objectives are to minimize workflow execution cost and time simultaneously, so two fitness functions are considered, F_{time} and F_{cost}. F_{time} corresponds to the workflow processing time and F_{cost} to the workflow execution cost. These functions have two parts, objective and penalty. Objective functions encourage the algorithm towards better solutions, while penalty functions degrade solutions that

Algorithm 1 (LWSGA Algorithm)

Input: N **(population size)**
G **(maximum number of generations)**
Output: A **(non-dominated solution set)**

Step 1. Population Initialization: Divide the workflow into levels and select the first population based on the problem constraints (Deadline and Budget) in a random manner.
Step 2. Non-Dominated Sort and Crowding Distance: The population is sorted based on non-domination and the crowding distance is calculated based on the method in [37].
Step 3. Selection: The mating pool is filled using the binary tournament selection method.
Step 4. Genetic Operations: Crossover and mutation operators are randomly selected and used on parent chromosomes in the mating pool to generate N offspring.
Step 5. New Population Generation: Combine the current solution and the new offspring to create an intermediate population of size $2N$ and select N of these based on front and crowding distance values.
Step 6. Repeat steps 2 to 5 until G iterations have been completed.

do not meet workflow constraints so they have less chance of being selected for the next population. The objective functions for an individual I are defined as follows

$$\text{Cost objective function}: f_{cost}(I) = {}^{cost(I)}/_{\beta} \tag{6}$$

$$\text{Time objective function}: f_{time}(I) = {}^{time(I)}/_{\mathcal{D}} \tag{7}$$

Considering workflow constraints, the penalty function for an individual I is defined as

$$P(I) = P_{budget}(I) + P_{deadline}(I) \tag{8}$$

where $P_{budget}(I)$ and $P_{deadline}(I)$ are the penalties for violating the budget and deadline, respectively, given by

$$P_{budget}(I) = \begin{cases} f_{cost}(I) & if(cost(I)+\omega)>\beta, \\ 0 & otherwise. \end{cases} \tag{9}$$

$$P_{deadline}(I) = \begin{cases} f_{time}(I) & if(time(I)+\alpha)>\mathcal{D}, \\ 0 & otherwise. \end{cases} \tag{10}$$

where ω and α are the acceptable deviations from the user budget and deadline, respectively.

A solution is feasible if it meets both budget and deadline constraints, so $P(I)$ is added to both objective functions to penalize solutions that are not feasible. However, this may eliminate solutions near the constraint boundaries. Thus solutions close to the deadline such as y_1 in Fig. 3, and close to the budget, such as y_2, are also used by the genetic operators as they can lead to feasible solutions. To achieve this, the penalty functions in (9) and (10) are set to 0 for small regions ω and α beyond the limits, respectively. However, at the end of the algorithm those solutions that do not meet the deadline or budget are eliminated from the final solution set.

The fitness functions for the cost and time are given by

$$F_{cost}(I) = f_{cost}(I) + P(I) \tag{11}$$

$$F_{time}(I) = f_{time}(I) + P(I) \tag{12}$$

respectively. A lower function value for a chromosome indicates better workflow scheduling.

3.4 Genetic Operators

Genetic operators are applied on population members called parents to generate new individuals called offspring. Three types of each of the genetic operators crossover and mutation are employed. The use of genetic operators can change the order of tasks, so dependencies may be violated. Thus task dependencies must be checked for all offspring. To avoid this problem, the offspring obtained with the proposed

Fig. 2 The chromosome structure (EO and SM fields) corresponding to Fig. 1

Levels	1	2			3		4		5
EO	T_{entry}	T1	T2	T0	T3	T4	T6	T5	T_{exit}
SM	Null	S1	S5	S2	S3	S2	S1	S5	Null

Algorithm 2 (Population Initialization)

Input: N **(population size)**
Output: B **(solution set)**

Step 1. Repeat steps 2 to 8 until the initial population is filled.
Step 2. For each of the L levels in the workflow do steps 3 to 8.
Step 3. For each task T in level l do steps 4 to 8 (tasks are selected randomly).
Step 4. Insert T_i in an appropriate EO cell.
Step 5. Compute the time until T_i is ready to be executed.
$\boldsymbol{PrepareTime}(\boldsymbol{T_i}) = \max_{T \in P} \boldsymbol{EndTime}\ (\boldsymbol{P_i})$, where P_i is the parent tasks for T_i.
Step 6. Select a service S_i^j randomly from the services available to execute T_i, and insert this service in the SM cell corresponding to T_i.
Step 7. Compute the data transfer time between T_i and its parents, $\boldsymbol{TransferTime}(\boldsymbol{T_i})$.
Step 8. Assign the free time slots S_i^j so that they start after $\boldsymbol{StartTime}(\boldsymbol{T_i})$, where
$$\boldsymbol{StartTime}(\boldsymbol{T_i}) = \boldsymbol{PrepareTime}(\boldsymbol{T_i}) + \boldsymbol{TransferTime}(\boldsymbol{T_i}). \quad (6)$$

operators are guaranteed to satisfy the task dependencies. Since the parents chosen from the current population will have a significant impact on the quality of the offspring, the Binary-Tournament Selection (BTS) method is used to select parents for the mating pool. This is a well-known mechanism which randomly selects pairs of individuals and chooses the one with the higher rank (lower fitness value) for the mating pool.

3.4.1 Crossover Operator

The crossover operator is used to create new entities by modifying parts of current individuals. The idea is that combining good solutions is likely to produce better solutions [44]. To increase the efficiency of the algorithm in different conditions and workflow structures, three crossover operators have been developed, and they are described below.

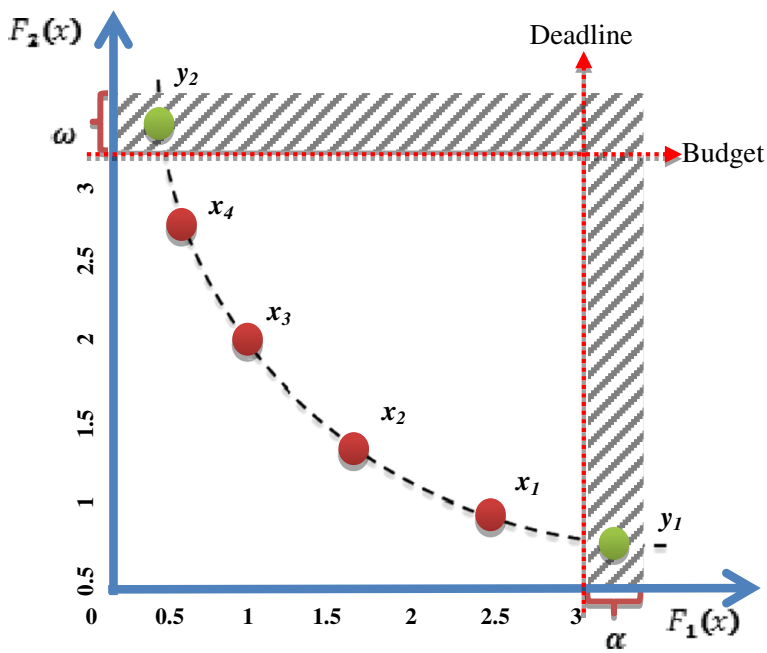


Fig. 3 The effect of ω and α on the penalty functions

I. Level exchange: This operator first generates a number between one and L randomly, where L is the number of workflow levels. Then, these levels are swapped between the parents. Figure 4 shows the result of this operator for the workflow in Fig. 1 where the third level is exchanged.

II. Task Exchange: This operator randomly selects a task from a level of the first parent, and according to the position of this task in the second parent, the tasks and services in these positions are exchanged within the parents. This is done for all workflow levels. The task exchange operator algorithm is as follows:

1. For each level l in list L
2. T_id = a random task from l.
3. P1= position of T_id in Parent 1.
4. P2= position of T_id in Parent 2.
5. Exchange tasks/services in P1 with tasks/services in P2 of Parent 1.
6. Exchange tasks/services in P1 with tasks/services in P2 of Parent 2.
7. End

Figure 5 show the task exchange operator when task T0 is selected in level 2. The position of T0 in Parent 1 is 3 and in Parent 2 is 1. Therefore tasks T0 and T1 in Parent 1, and tasks T2 and T0 in Parent 2, and their corresponding services, are exchanged.

III. Service Exchange: This operator selects one task from each level of the workflow randomly, and exchanges the corresponding services between parents. The algorithm of this operator is as follows:

1. For each level l in list L
2. T_id = a random task from l.
3. P1= position of T_id in Parent 1.
4. P2= position of T_id in Parent 2.
5. Exchange service in P1 of Parent 1 with service in P2 of Parent 2.
6. End

Figure 6 shows the service exchange operator when task T2 in level 2 of the workflow is selected. The position of task T2 in Parent 1 is 2 and in Parent 2 is 3, so service S5 in Parent 1 is exchanged with service S4 in Parent 2.

3.4.2 Mutation Operator

The mutation operator is used to allow offspring to obtain features that their parents do not have. This operator helps the algorithm move from local optimums and search in new objective spaces. Three types of mutation operators have been developed, and they are described below.

I. Exchange Task: This operator exchanges the positions of two tasks in one parent. The selected tasks should be in same level to meet the task dependencies. The steps of this operator are as follows:

1. Select a random workflow level l.
2. Select two random tasks P1 and P2 from level l.
3. Exchange P1 and P2.

II. Exchange Service: This operator changes the services for up to $N/4$ workflow tasks. The new services are chosen randomly. The steps of this operator are as follows:

1. Generate a random number K between 1 and $N/4$.
2. Repeat steps 3 to 5 K times.
3. Randomly select a task T from the parent.
4. Randomly select a service S from the available services for T.
5. Assign S to T and place in it the chromosome.

Figure 7 shows an example of the exchange service operator. In this case tasks T3 and T5 are randomly selected to change their services. The new services are randomly selected from the available services for the tasks. The available services for T3 are S1, S3, S5, and S8, and for T5 are S2, S5, S6, and S7. The current services are highlighted and the selected services are underlined.

III. Recorder Level: This operator selects one chromosome level and randomly changes the order of the tasks as well as assigning new services. Figure 8 shows an example of this operator where the second level is selected. All tasks in this level are reordered and new services are randomly selected from the list of available services. The available services for T0 are S1, S2, S5, and S8, for T1 are S1, S4, S6, S7, and for T2 are S2, S3, S5, and S8. The current services are highlighted and the selected services are underlined.

4 Performance Results

In order to evaluate the efficiency and performance of the workflow scheduling algorithms in a grid environment, four metrics are employed,

Before Crossover

Parent 1:

Level:	1	2			3		4		5
EO	T_{entry}	T0	T2	T1	T4	T3	T5	T6	T_{exit}
SM:	Null	S2	S5	S1	S2	S3	S5	S1	Null

Parent 2:

Level:	1	2			3		4		5
EO:	T_{entry}	T1	T2	T0	T3	T4	T5	T6	T_{exit}
SM:	Null	S4	S1	S2	S5	S4	S1	S3	Null

After Crossover

Offspring 1:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T2	T1	T3	T4	T5	T6	T_{exit}
SM:	Null	S2	S5	S1	S5	S4	S5	S1	Null

Offspring 2:

Level:	1	2			3		4		5
EO:	T_{entry}	T1	T2	T0	T4	T3	T5	T6	T_{exit}
SM:	Null	S4	S1	S2	S2	S3	S1	S3	Null

Fig. 4 An example of the level exchange crossover operation

the hypervolume indicator, the epsilon indicator, convergence, and a new metric which will be described later. The proposed LWSGA algorithm is compared with three well-known algorithms: SPEA2, PAES and NSGA-II. Next, the characteristics of the simulation environment are described, and then the performance results are presented and discussed.

4.1 Workflow Applications

Workflows have different structures which can affect the performance of scheduling algorithms. To evaluate these algorithms, two approaches are commonly employed to choose the workflows:

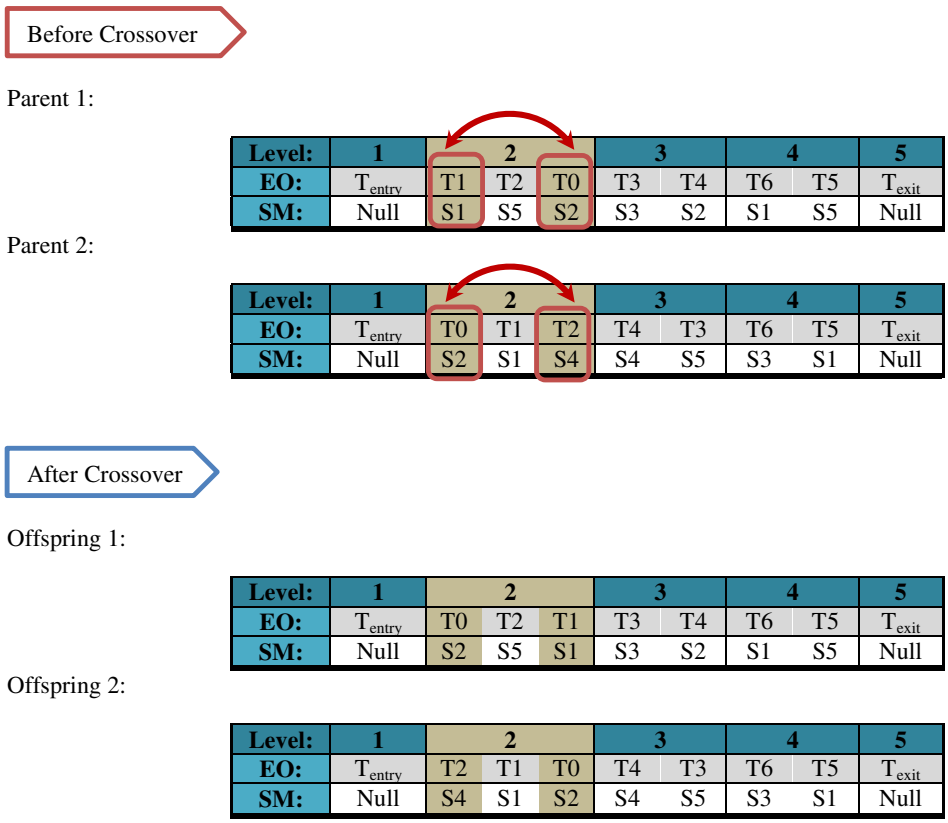
1. Randomly generate DAGs with different characteristics.
2. Select from realistic workflow libraries that are accepted by the scientific community.

The second approach seems more appropriate for the fair evaluation of workflow scheduling algorithms [45, 46], thus the workflows given by Bharathi et al. [47] are employed. This workflow library was compiled from diverse scientific applications. Three workflow types have been selected for use in this section: Montage (astronomy), Epigenomics (biology), and LIGO (gravitational physics). They are available in Directed Acyclic Graph in XML (DAX) format on their web site.[1] Figure 9 shows the structure of a small instance of each workflow. They have structural differences in terms of basic components (pipeline, data distribution, etc.) and composition. In this paper, three different workflow sizes are considered: small (about 25 tasks), medium (about 50 tasks), and large (about 100 tasks).

4.2 Simulation Environment

The experiments were carried out by simulation using an implementation in MATLAB. All algorithms, grid resources and connections, scheduling, and their specific characteristics were defined and implemented in this environment. The performance results were then evaluated using the metrics implemented in MATLAB. To simulate resources in utility grids and execute the workflows, eight services with different

[1] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.



Level:	1	2			3		4		5
EO:	T_{entry}	T1	T2	T0	T3	T4	T6	T5	T_{exit}
SM:	Null	S1	S5	S2	S3	S2	S1	S5	Null

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T1	T2	T4	T3	T6	T5	T_{exit}
SM:	Null	S2	S1	S4	S4	S5	S3	S1	Null

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T2	T1	T3	T4	T6	T5	T_{exit}
SM:	Null	S2	S5	S1	S3	S2	S1	S5	Null

Level:	1	2			3		4		5
EO:	T_{entry}	T2	T1	T0	T4	T3	T6	T5	T_{exit}
SM:	Null	S4	S1	S2	S4	S5	S3	S1	Null

Fig. 5 An example of the task exchange crossover operation

properties and processing speeds are considered. For each task of a workflow, only those services able to execute the task are selected. It is assumed that each task has at most five services that can execute it. This assumption is realistic due to the diversity of task requirements in a grid environment. For example, of the five services S1, S1, S3, S4, and S5 available for a workflow, Task 3 can only be executed on services S2, S3, and S5. The processing cost must be considered for each service, and in general the resources with higher speed are more costly. Each task is assigned to a free time slot of one service which is able to execute it. The execution time and cost of each task on each service can be predicted using parameters such as task length and the number of instructions a service can execute per unit time. For example, Table 1 shows the execution time and cost of Task 3 on each service provider. This indicates that Task 3 can be executed faster on service S3, but the cost is higher.

A task may create data for its children. This data must be transmitted to the corresponding service provider before the children can start execution. It is assumed that the connections between service providers have bandwidths of 100, 200, 512, and 1024 megabytes per second, and these are assigned to services randomly. The data transfer time and cost can then be predicted from the available bandwidth and the amount of data to be transmitted. An increased bandwidth will typically lower the data transfer time but increase the cost.

The performance of the algorithms is investigated under three constraint environments denoted as relaxed, medium and tight. In the relaxed environment, users have high budget and deadline values, whereas in the tight constraint environment these values are low. The budgets and deadlines for each environment are defined by minimum and maximum values. T_{max} is the maximum workflow execution time with minimum cost C_{min}, and C_{max} is the maximum workflow execution cost with minimum execution time T_{min}. In this paper, C_{max} and C_{min} were obtained using the workflow execution

Before Crossover

Parent 1:

Level:	1	2			3		4		5
EO:	T_{entry}	T1	T2	T0	T3	T4	T6	T5	T_{exit}
SM:	Null	S1	S5	S2	S3	S2	S1	S5	Null

Parent 2:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T1	T2	T4	T3	T6	T5	T_{exit}
SM:	Null	S2	S1	S4	S4	S5	S3	S1	Null

After Crossover

Offspring 1:

Level:	1	2			3		4		5
EO:	T_{entry}	T1	T2	T0	T3	T4	T6	T5	T_{exit}
SM:	Null	S1	S4	S2	S3	S2	S1	S5	Null

Offspring 1:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T1	T2	T4	T3	T6	T5	T_{exit}
SM:	Null	S2	S1	S5	S4	S5	S3	S1	Null

Fig. 6 An example of the service exchange crossover operation

cost optimization algorithm GreedyCost [4], and $T_{\min}$ and $C_{\max}$ were obtained using the workflow execution time optimization algorithm HEFT [20]. The workflow execution deadline and budget are then given by

$$\mathcal{D} = T_{\max} - k\,(T_{\max} - T_{\min}) \tag{13}$$

$$\beta = C_{\max} - k\,(C_{\max} - C_{\min}) \tag{14}$$

respectively, where k is 0.2, 0.5 and 0.8 for the relaxed, medium and tight constraint environments, respectively. The genetic algorithm parameters are given in Table 2.

Before Mutation

Parent:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T2	T1	T4	T3	T5	T6	T_{exit}
SM:	Null	S2	S5	S1	S2	S3	S5	S1	Null

After Mutation

Offspring:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T2	T1	T4	T3	T5	T6	T_{exit}
SM:	Null	S2	S5	S1	S2	S8	S6	S1	Null

Fig. 7 An example of the exchange service mutation operation

Before Mutation

Parent:

Level:	1	2			3		4		5
EO:	T_{entry}	T0	T2	T1	T4	T3	T5	T6	T_{exit}
SM:	Null	S2	S5	S1	S2	S3	S5	S1	Null

After Mutation

Offspring:

Level:	1	2			3		4		5
EO:	T_{entry}	T2	T0	T1	T4	T3	T5	T6	T_{exit}
SM:	Null	S2	S8	S4	S2	S8	S6	S1	Null

Fig. 8 An example of the reorder level mutation operation

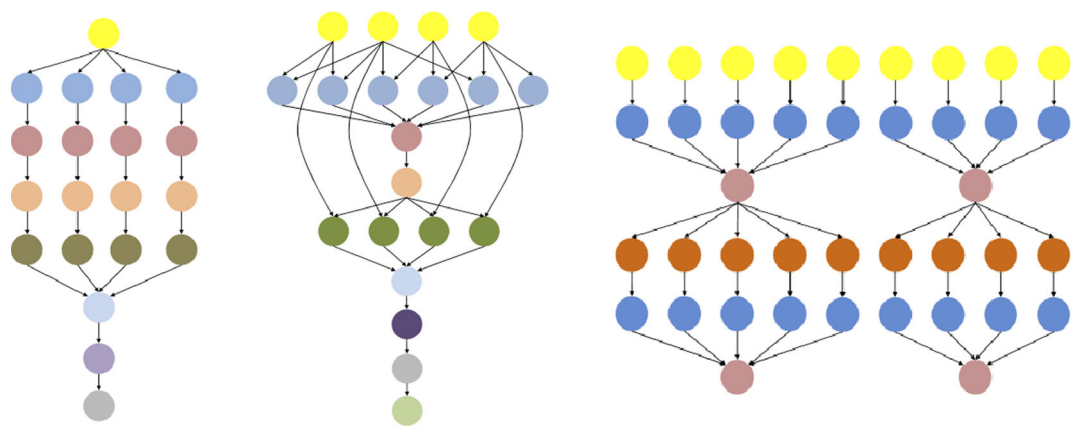
4.3 Performance Measures

To evaluate the performance of the algorithms, the solution sets obtained are evaluated using the Hypervolume indicator [48, 51], epsilon indicator [49], convergence metric [50], and the time required to schedule the workflow (the new metric). The corresponding results are presented below.

4.3.1 Hypervolume Indicator

The hypervolume indicator,I_H, is used to evaluate results from genetic algorithms. This metric gives the volume of the solution space covered by a non-dominated Pareto front returned by an algorithm, and how close the solutions are to optimal. The algorithms were repeated 20 times for each of the workflow and constraint combinations. A reference set R was then constructed by merging all of the resulting non-dominated fronts. Then, the hypervolume difference indicator I_H^- [48] was used to measure the difference between the non-dominated fronts obtained from each combination for an algorithm and R. A lower value of I_H^- indicates better algorithm performance [48].

Figure 10 a-1, b-1, and c-1 show the best non-dominated front obtained from each of the algorithms for the workflows and constraints. The x axis shows the workflow execution time and the y axis shows the workflow execution cost, both normalized to the range [0, 1]. Figure 10 a-2, b-2, and c-2 show the corresponding values of I_H^- according to the results



(a) Epigenomics (b) Montage (c) LIGO

Fig. 9 The structure of three realistic scientific workflows [47]

Table 1 Execution time and cost of Task 3 on five services

Service	S1	S2	S3	S4	S5
Time (seconds)	∞	300	100	∞	400
Cost (dollars)	∞	80	200	∞	50

from 20 runs. The x axis indicates the algorithm employed and the$y$ axis the I_H^- value. These results indicate that the LWSGA algorithm provides better non-dominated fronts, except for the large Montage workflow with medium and tight constraints. However, in these cases the other algorithms obtained better solutions for only a few locations in the solution space, and I_H^- for the LWSGA algorithm was only slightly higher. In medium LIGO workflows with tight constraints, the I_H^- values for the SPEA2, PAES and NSGA-II algorithms have the same functionality but LWSGA provides better results. In addition, in large Epigenomics workflows with relaxed constraints, the I_H^- values for LWSGA are much less than NSGA-II, PAES and SPEA2. With medium Montage workflows and medium constraints, the I_H^- values for all algorithms are similar and the performance differences are minimal.

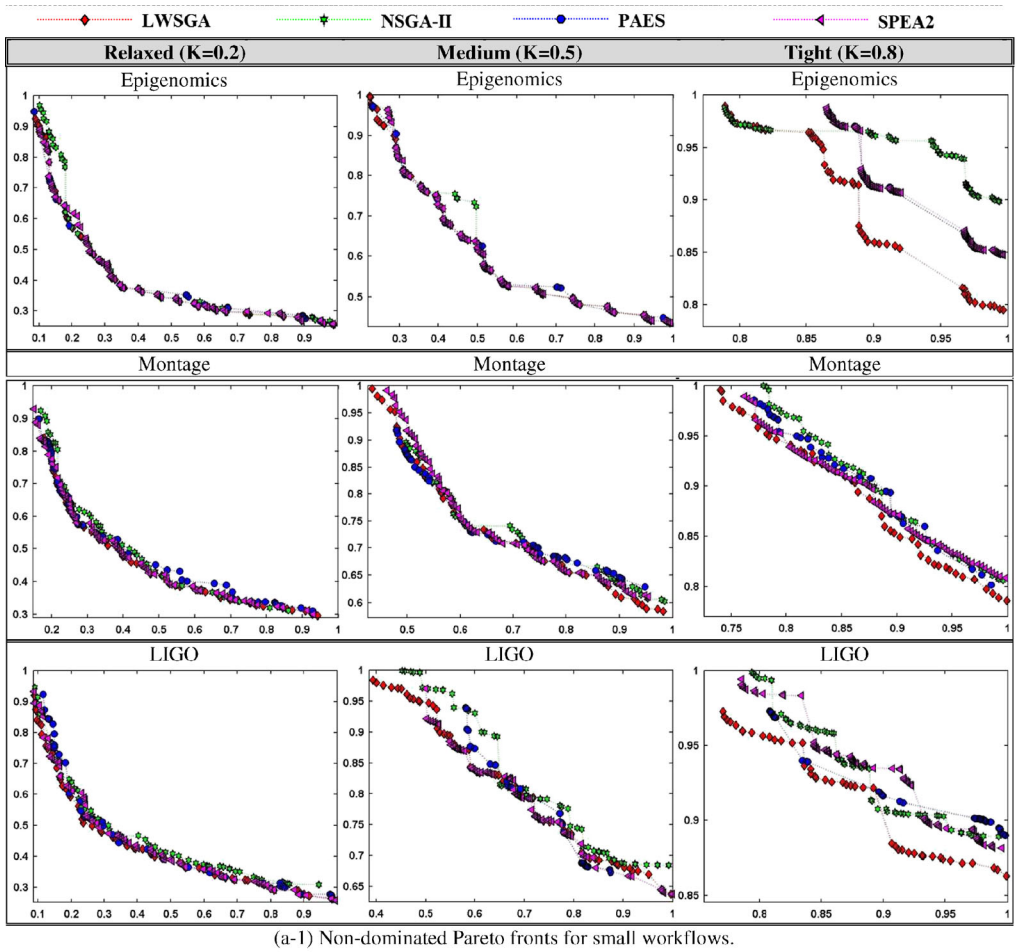
In addition to providing better solutions, the proposed algorithm has better coverage of the solution space than the other algorithms. In particular, the coverage of the solution space close to the constraint boundaries is much better. For example, Fig. 10 c-1 shows the results for large LIGO workflows with relaxed constraints. In this case, the LWSGA algorithm covers the entire solution space, but the solutions obtained with the other algorithms are clustered near the center of the solution space with none near the boundaries.

4.3.2 Epsilon Indicator

Although I_H^- is widely used to compare GA performance, it sometimes provides differing results for two solution distributions even when they are similar [31]. The epsilon indicator, $I_{\varepsilon+}$, is used to make more specific comparisons. According to the degree of non-domination ε, it specifies how much two solution distributions differ. A lower value of $I_{\varepsilon+}$ indicates a better front. $I_{\varepsilon+}$ in this section was obtained using the same simulation parameters as for I_H. Each of the algorithms was run 20 times and the average used to measure $I_{\varepsilon+}$. The results obtained are presented in Fig. 11, where the x axis indicates the algorithm and the y axis gives the corresponding values of $I_{\varepsilon+}$. This shows that $I_{\varepsilon+}$ for the LWSGA algorithm is in most cases significantly lower than for the other algorithms. Thus the differences between the non-dominated solutions generated by the proposed algorithm and the reference set R are lower. However, with small Montage workflows and relaxed constraints, the SPEA2 algorithm has the lowest $I_{\varepsilon+}$ values, while with large Montage workflows and relaxed constraints, the PAES algorithm has the lowest $I_{\varepsilon+}$ values. However, in many cases the non-dominated fronts obtained are not significantly different. For example, with small LIGO workflows and tight constraints, the $I_{\varepsilon+}$ values are essentially the same. Note that algorithm performance varies according to the test

Table 2 The genetic algorithm parameters

Parameter	Type/Value
Population Size	70 individuals
Crossover probability	0.7
Mutation Probability	0.3
Fitness Function	time and cost
Initial Population	random generation
External Archive Size (SPEA2, PAES)	70 individuals
Maximum Number of Generations (NSGA-II, SPEA2, LWSGA)	200 generations
Maximum Number of Iterations (PAES)	20000 iterations



(a-1) Non-dominated Pareto fronts for small workflows.

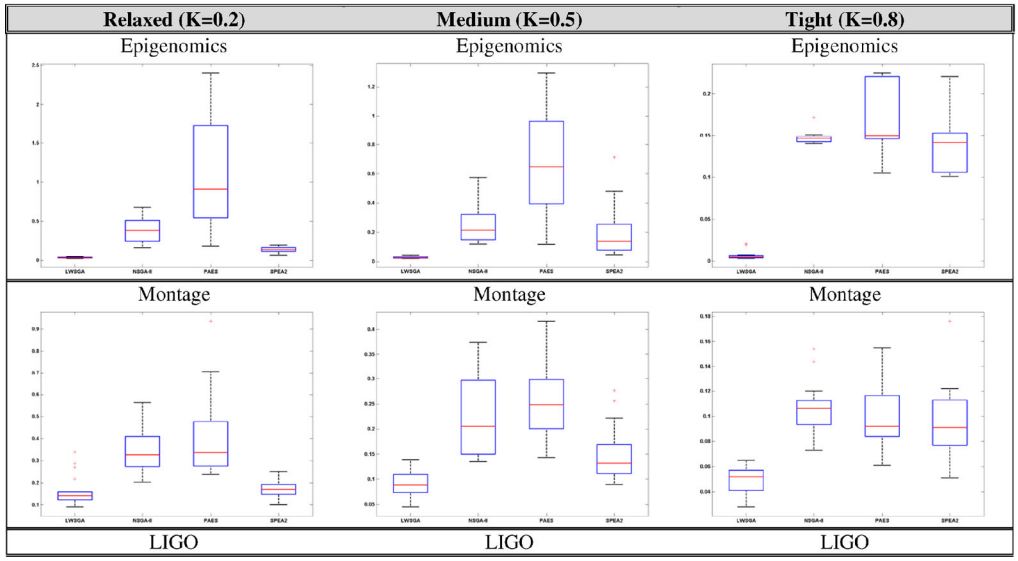
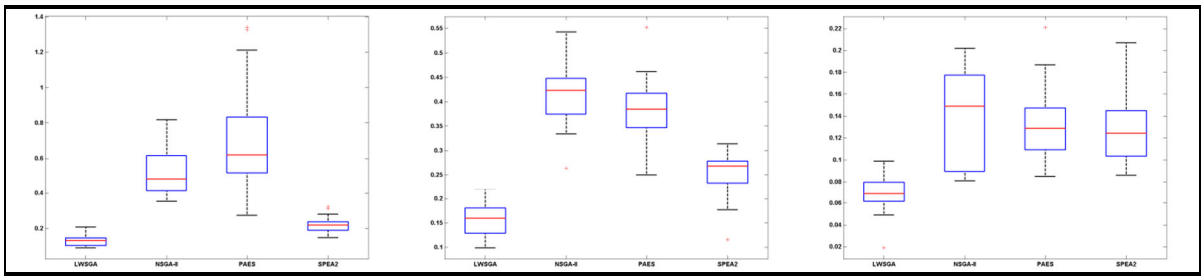
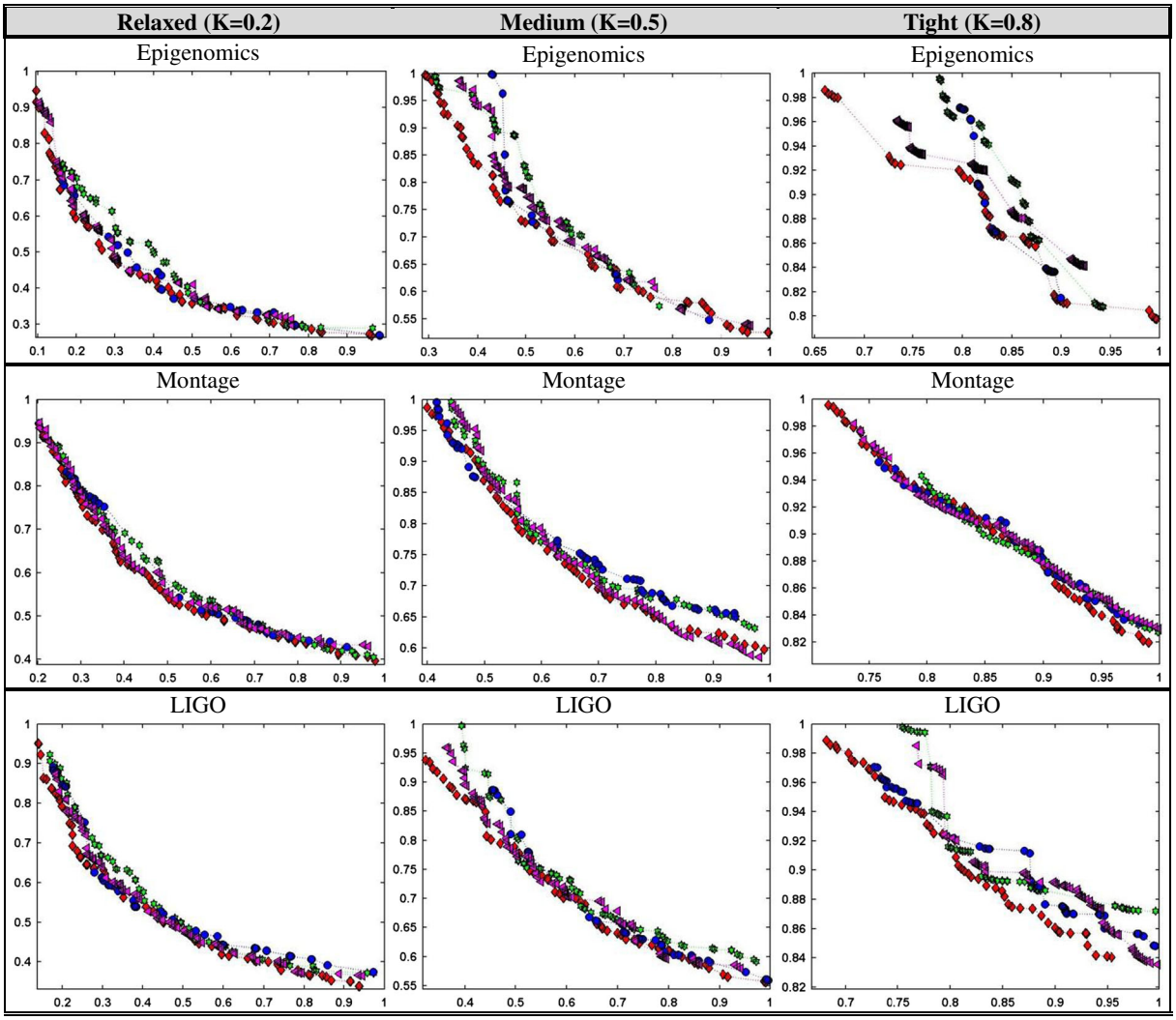


Fig. 10 Non-dominated fronts and the hypervolume indicators (box plots) for workflows under different constraints: **a**-1, **b**-1, and **c**-1 non-dominated front solutions for small, medium, and large workflows, respectively; **a**-2, **b**-2, and **c**-2 hypervolume indicators for small, medium, and large workflows, respectively



(a-2) Values of I_H^- for small workflows.

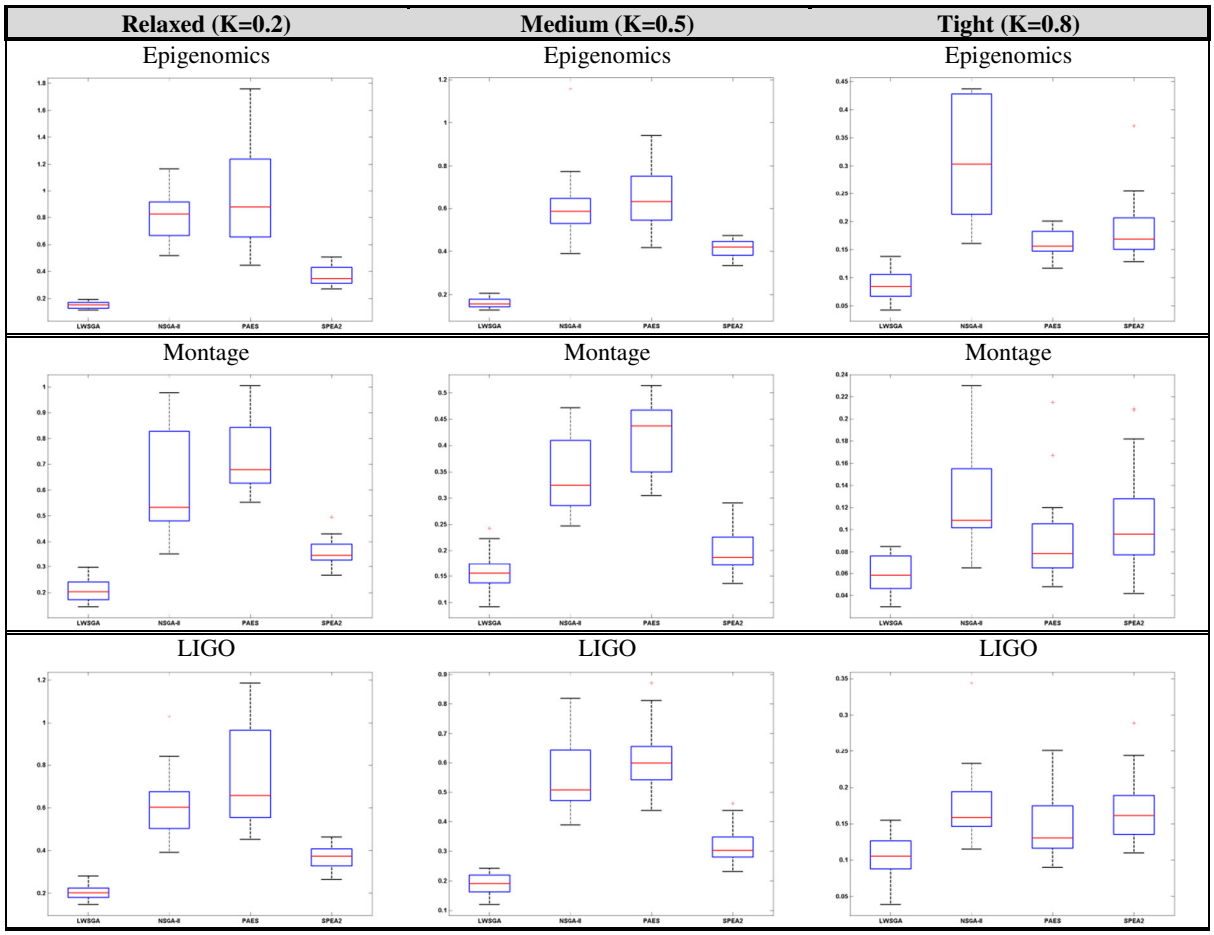


(b-1) Non-dominated Pareto fronts for medium workflows.

Fig. 10 (continued)

case. For example, in small Montage workflows with relaxed constraints, SPEA2 has a lower $I_{\varepsilon+}$ value than PAES, but in large Montage workflows the results are reversed.

In medium Epigenomics workflows with medium constraints, the $I_{\varepsilon+}$ values obtained for LWSGA are significantly different than with the others algorithms, while SPEA2, PAES and NSGA-II have similar



(b-2) Values of I_H^- for medium workflows.

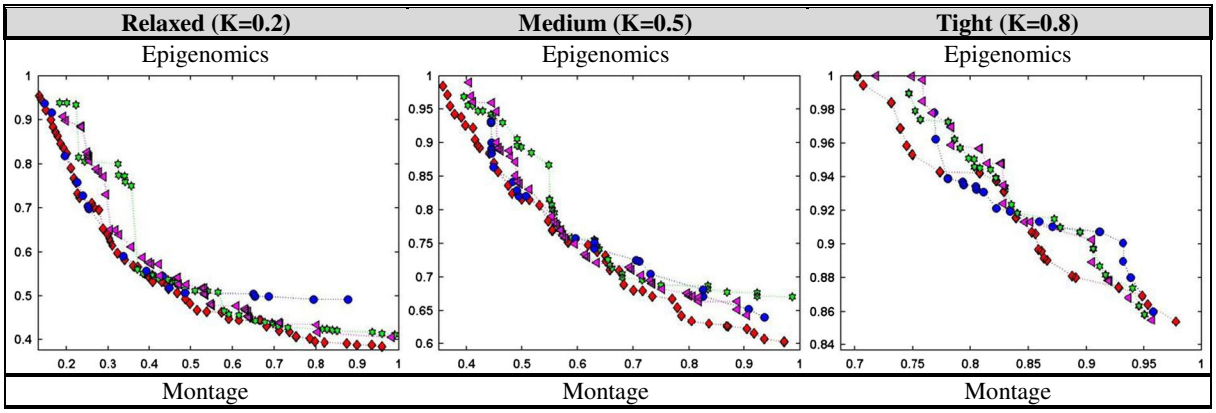
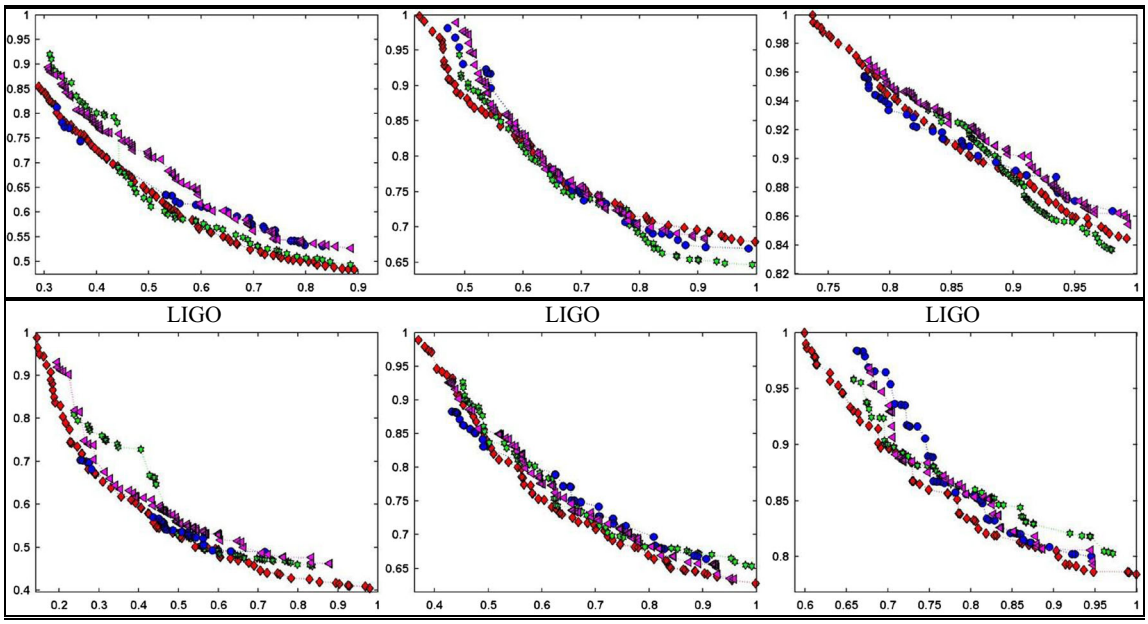
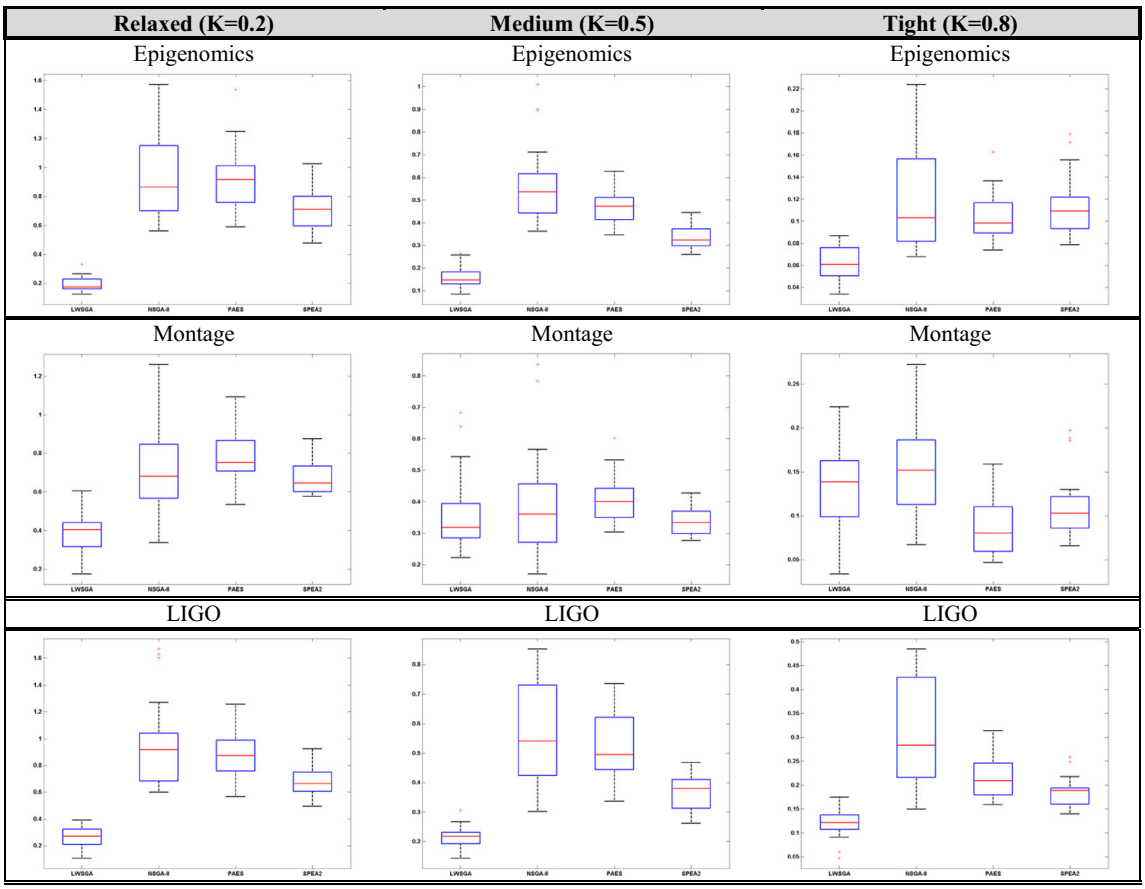


Fig. 10 (continued)

values. Moreover, the $I_{\varepsilon+}$ values in medium Epigenomics workflows with tight constraints for SPEA2 and PAES are similar, but both are better than NSGA-II and worse than LWSGA. In large LIGO workflows with tight constraints, the $I_{\varepsilon+}$ values for LWSGA are much lower than NSGA-II and PAES, while

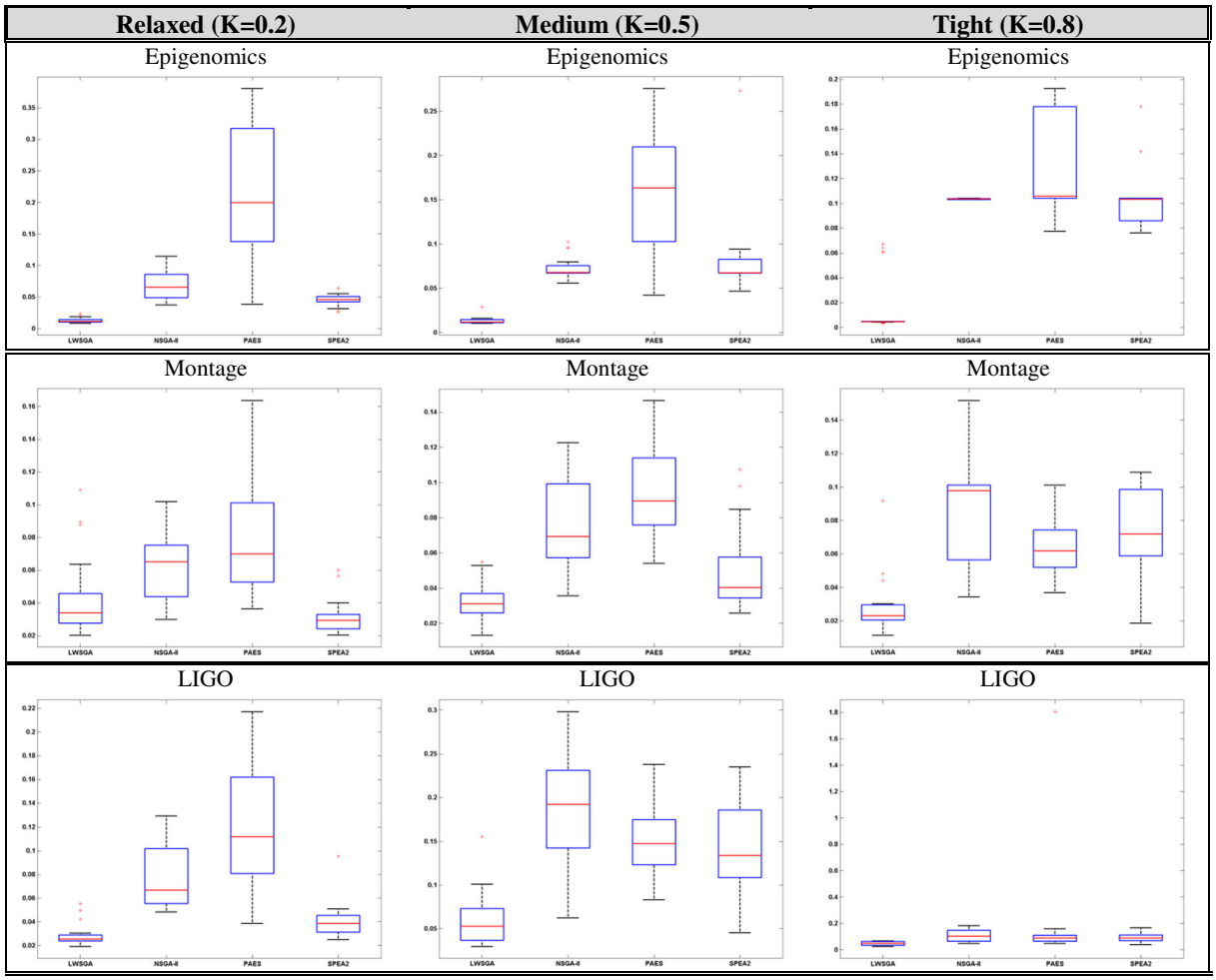


(c-1) Non-dominated Pareto fronts for large workflows.



(c-2) Values of $I_{\overline{H}}$ for large workflows.

Fig. 10 (continued)



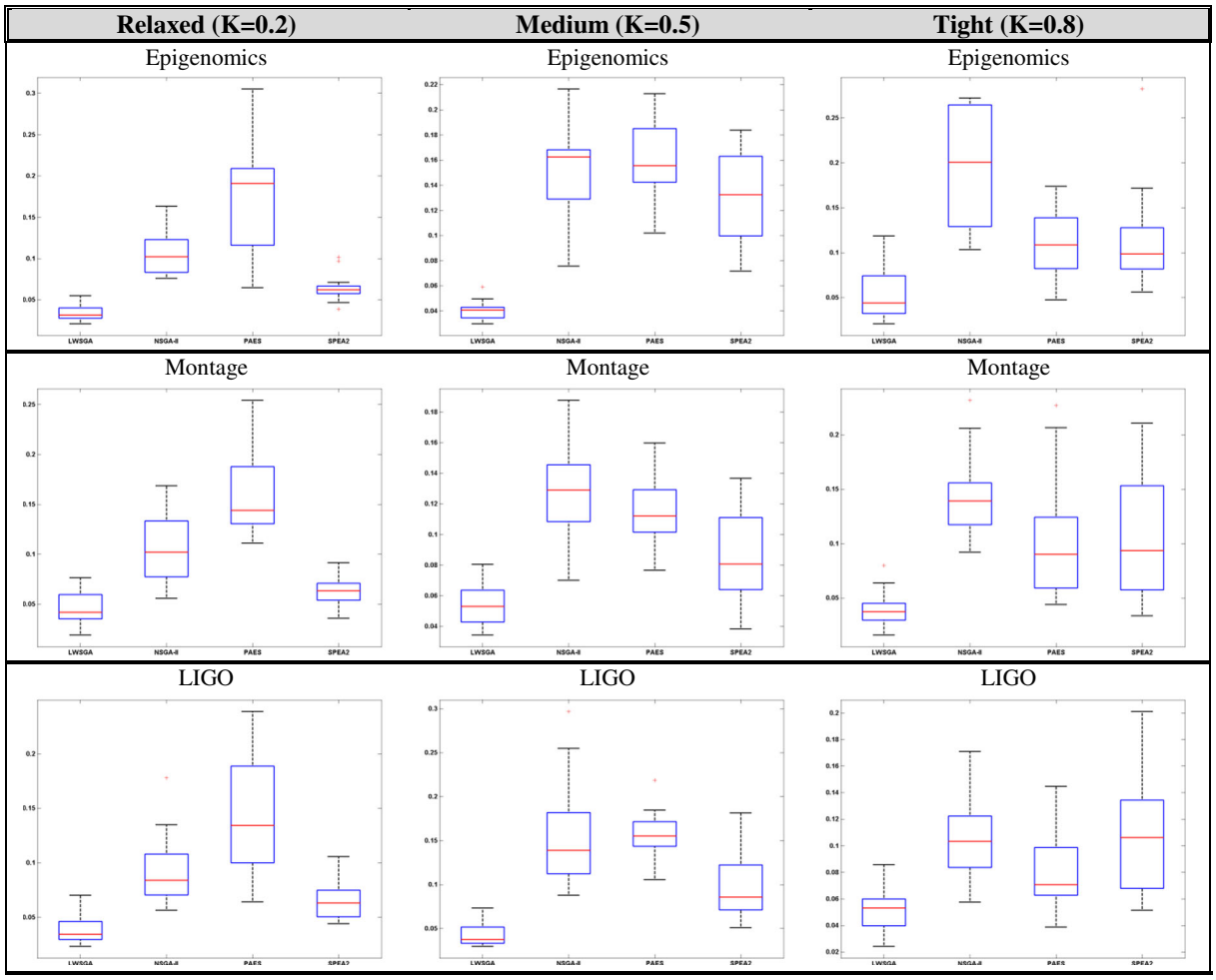
(a)

Fig. 11 The epsilon metric for different workflows and constraints, (**a**) small workflows, (**b**) medium workflows, and (**c**) large workflows

SPEA2 has similar performance. With medium Montage workflows and relaxed constraints, the $I_{\varepsilon+}$ values for LWAGA and SPEA2 are similar and better than NSGA-II, while PAES has the worst performance. The $I_{\varepsilon+}$ values in medium LIGO workflows with tight constraints for NSGA-II and PAES are somewhat the same, but neither is better than SPEA2 and LWSGA. In Epigenomics workflows, NSGA-II is sometimes better than both SPEA2 and PAES, but in other cases is worse. Thus it is difficult to choose a best algorithm. As another example, for LIGO workflows, in most cases SPEA2 is better than PAES and NSGA-II, but not always. Therefore, additional results using other metrics are needed to definitively rank the algorithms.

4.3.3 Convergence Metric

Multi-objective optimization has two goals: (1) keep solutions in the solution space, and (2) converge towards the Pareto-optimal set. The first goal is measured adequately by the hypervolume and epsilon metrics. In this section, the convergence metric, γ [50], is used to measure the convergence of the non-dominated fronts obtained in the Pareto-optimal sets. The same simulation parameters and environments were used as in the previous sections. Each of the algorithms was run 20 times for each case. A lower γ value indicates better convergence towards the Pareto-optimal set. The mean ($\overline{\gamma}$) and variance ($\sigma\gamma$) of this metric are shown in Table 3 for different



(b)

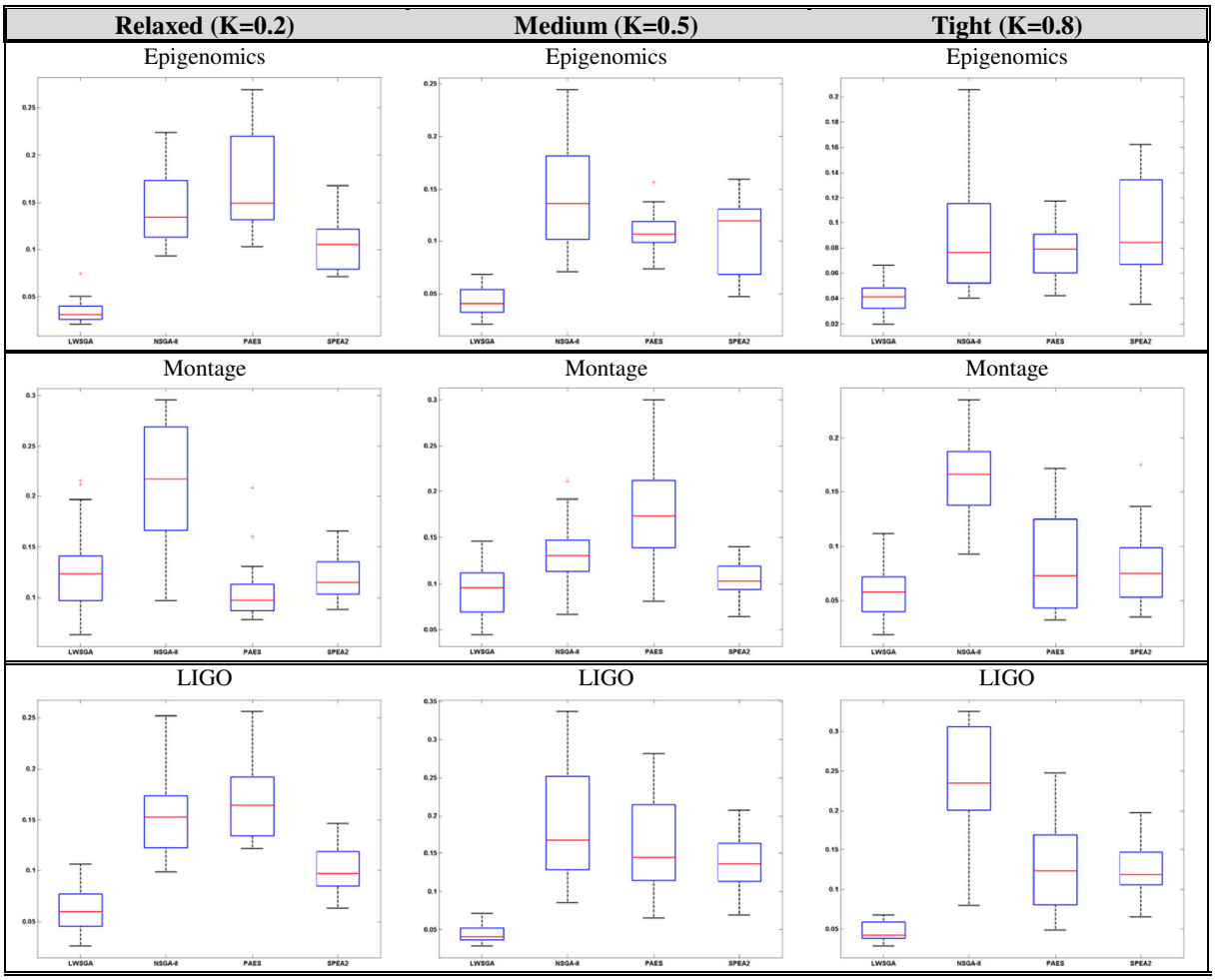
Fig. 11 (continued)

workflows and constraint environments. This shows that in most cases the LWSGA algorithm is significantly better than the others as it has lower means and variances. For example, with the Epigenomics workflows, this algorithm has considerably lower values. However, with large Montage workflows and tight constraints the other algorithms have lower $\overline{\gamma}$ values than LWSGA, and with small LIGO workflows and tight constraints, PAES has the lowest values. In many cases, the difference between the algorithms is small.

4.3.4 The Proposed Metric (Stability Metric)

In addition to convergence metric, it is important to determine how the algorithms converge to the final solution set and at what point they become stable. To quantify these parameters, a new metric is proposed called the stability metric, $\hat{\gamma}$. To calculate $\hat{\gamma}$ $\hat{\gamma}$, after each generation of the genetic algorithm, γ is calculated. This is done for multiple runs of each algorithm. The mean of these values gives the $\hat{\gamma}$ metric for each generation. A lower mean denotes better algorithm performance. This metric indicates how many GA generations are required to achieve a stable solution with an acceptable population.

The $\hat{\gamma}$ metric was determined using the same simulation environment and parameters as used previously. Each of the algorithms was run 20 times, and the results are shown in Fig. 12. The x axis shows the generation number and the y axis the metric values. This shows that the LWSGA algorithm becomes stable much faster than the other algorithms



(c)

Fig. 11 (continued)

with relaxed constraints, except with the large Montage workflows where NSGA-II has the lowest $\hat{e}$. In the other cases, $\hat{e}$ for the LWSGA algorithm is as good or better, except with large Epigenomics workflows and tight constraints where NSGA-II again has the lowest $\hat{e}$ value. The number of generations for the LWSGA, NSGA-II, and SPEA2 algorithms was 200 and the number of iterations for PAES was 20000. Thus with PAES, $\hat{e}$ was calculated only every 100 iterations as this is equivalent to one generation.

4.3.5 Scheduling Time

In a grid environment, algorithms should provide scheduling solutions quickly. Thus the goals are to improve the quality of the solutions generated and also reduce the time required for workflow scheduling. To compare scheduling times, each of the algorithms for all workflows and constraints was executed on the same computer. The number of generations for the LWSGA, SPEA2, and NSGA-II algorithms was 200 and 500, and the number of iterations for PAES was 20000 and 50000. The results are given in Table 4. This shows that in all cases the LWSGA algorithm is significantly faster than the other algorithms (and in most cases the quality of the solutions is better). It is about 6 to 12 times faster than the SPEA2 algorithm. This is due to the time required to update the external population and the complex operations within the algorithm. Because the LWSGA algorithm does not need to check task dependencies, it is faster than the NSGA-II algorithm. Further, it is

Table 3 The mean ($\overline{\gamma}$) and variance ($\sigma\gamma$) of the convergence metric for (a) small workflows, (b) medium workflows, and (c) large workflows

Size: Small	Metrics	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	$\overline{\Upsilon}$	0.0026	0.0066	0.026	0.0079	0.018	0.023	0.0096	0.024	0.034
	σ_Υ	0.0	0.0	0.000082	0.0	0.0	0.0	0.0	0.000030	0.000050
NSGA-II	$\overline{\Upsilon}$	0.023	0.015	0.045	0.022	0.025	0.025	0.034	0.048	0.042
	σ_Υ	0.000075	0.000090	0.000017	0.000046	0.00014	0.000025	0.000050	0.00011	0.00033
PAES	$\overline{\Upsilon}$	0.0077	0.013	0.069	0.021	0.024	0.029	0.028	0.048	0.15
	σ_Υ	0.000018	0.00014	0.0013	0.000029	0.000026	0.000085	0.000086	0.00010	0.24
SPEA2	$\overline{\Upsilon}$	0.0084	0.0096	0.049	0.012	0.018	0.022	0.014	0.029	0.038
	σ_Υ	0.0	0.000070	0.00091	0.0	0.000047	0.00010	0.0	0.000030	0.00015

(a)

Size: Medium	Metrics	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	$\overline{\Upsilon}$	0.011	0.020	0.033	0.013	0.024	0.023	0.015	0.026	0.034
	σ_Υ	0.0	0.0	0.000037	0.0	0.000011	0.000040	0.0	0.000010	0.000050
NSGA-II	$\overline{\Upsilon}$	0.051	0.058	0.087	0.042	0.035	0.027	0.045	0.051	0.042
	σ_Υ	0.00014	0.00024	0.0018	0.00012	0.000075	0.00023	0.000087	0.00022	0.00041
PAES	$\overline{\Upsilon}$	0.023	0.050	0.032	0.031	0.053	0.025	0.031	0.044	0.040
	σ_Υ	0.000033	0.00033	0.000060	0.000053	0.00010	0.000010	0.000031	0.000088	0.00016
SPEA2	$\overline{\Upsilon}$	0.027	0.035	0.043	0.026	0.023	0.028	0.027	0.029	0.042
	σ_Υ	0.000017	0.000028	0.00020	0.000010	0.000019	0.00014	0.000014	0.000020	0.00021

(b)

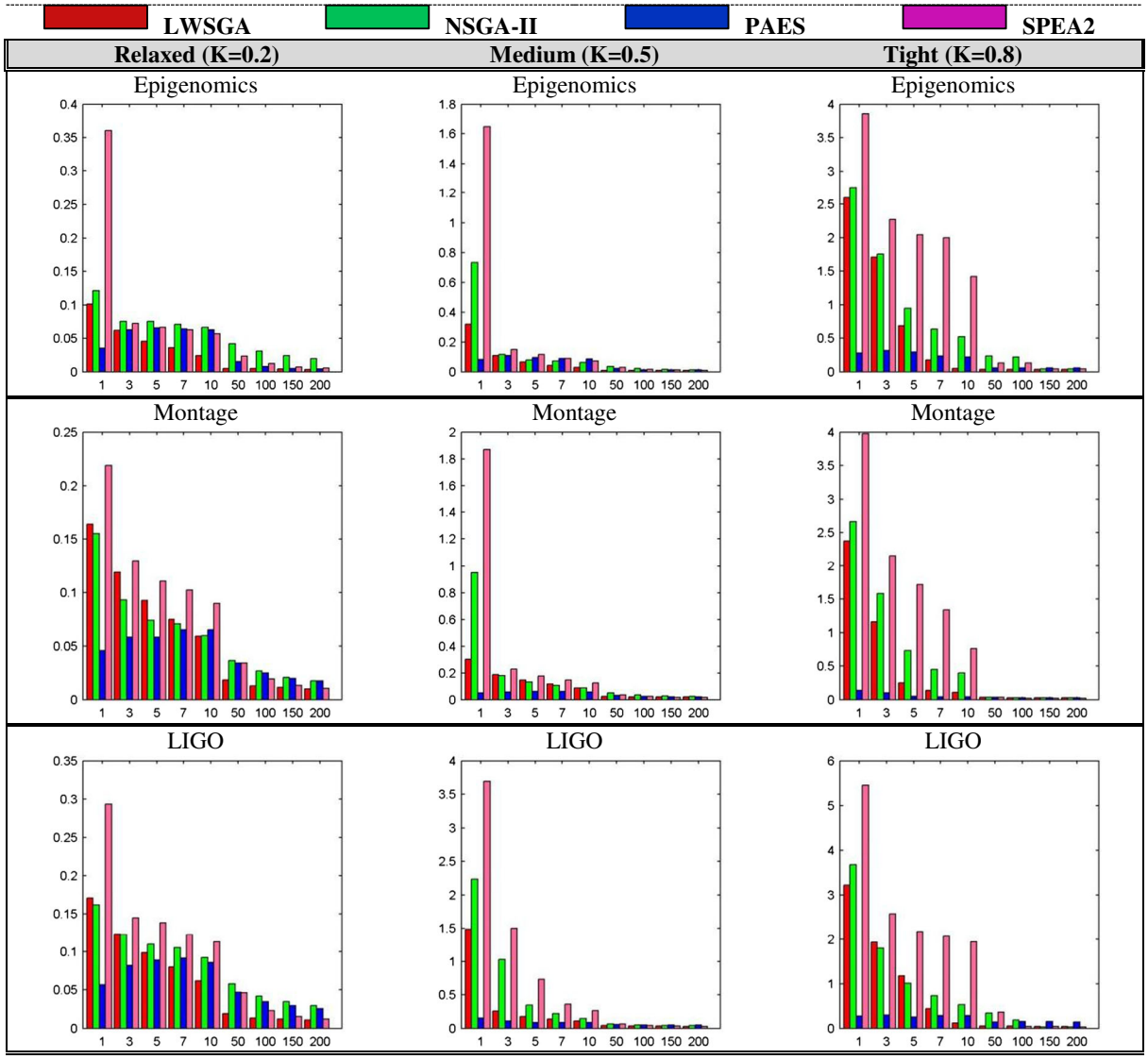
Size: Large	Metrics	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	$\overline{\Upsilon}$	0.015	0.021	0.026	0.023	0.030	0.036	0.016	0.0240	0.028
	σ_Υ	0.0	0.000012	0.0	0.000030	0.000030	0.000042	0.0	0.0	0.0
NSGA-II	$\overline{\Upsilon}$	0.051	0.061	0.029	0.053	0.033	0.029	0.062	0.050	0.029
	σ_Υ	0.00018	0.00098	0.00020	0.00051	0.00070	0.00074	0.00095	0.00049	0.00055
PAES	$\overline{\Upsilon}$	0.044	0.049	0.026	0.048	0.052	0.025	0.046	0.057	0.044
	σ_Υ	0.00016	0.00011	0.000094	0.00024	0.00015	0.000059	0.00021	0.00019	0.000080
SPEA2	$\overline{\Upsilon}$	0.048	0.032	0.025	0.057	0.039	0.031	0.053	0.035	0.030
	σ_Υ	0.000095	0.000054	0.000053	0.000090	0.000048	0.00014	0.00011	0.000047	0.0

(c)

2 to 8 times faster than the PAES algorithm. Note that in this paper, the number of PAES iterations is 100 times the number of generations with the other algorithms. This factor was chosen to obtain good solutions which are competitive with the other algorithms. The scheduling time in some cases increases dramatically as the workflow sizes increases. For example, with small Montage workflows and medium constraints, the NSGA-II algorithm requires 15 s, and with the large workflow this increases to 115 s. Thus while the size of the workflow has increased by a factor of 4, the scheduling time has increased by a factor of 7. Conversely, with the LWSGA algorithm the scheduling time increased by only a factor of 2.5. Changes in the workflow type or constraints did not have a significant impact on algorithm speed. Further, increasing the workflow size did not affect the performance of the LWSGA algorithm, but had a significant impact on the other algorithms.

4.4 Penalty Mechanisms

To determine the effects of the penalty mechanisms on the quality of the solutions, the LWSGA algorithm



(a) **Small Workflows**

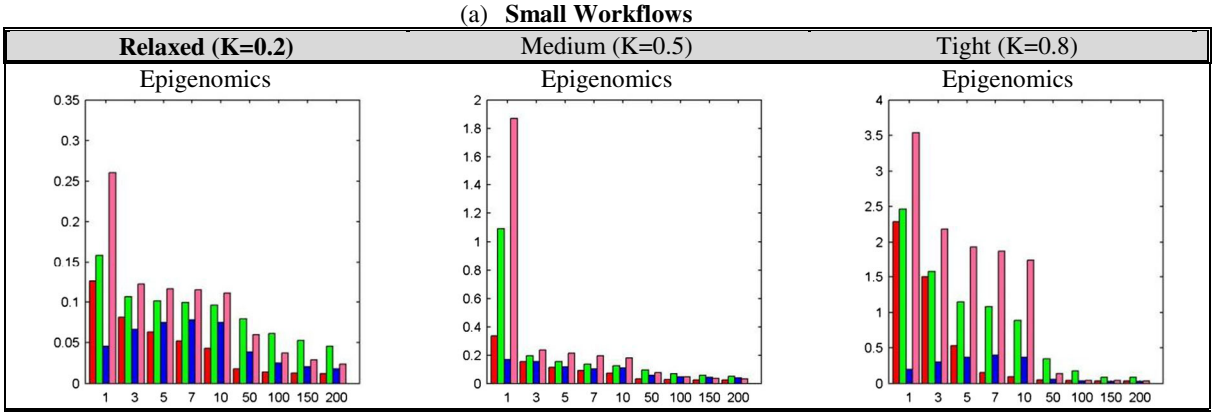
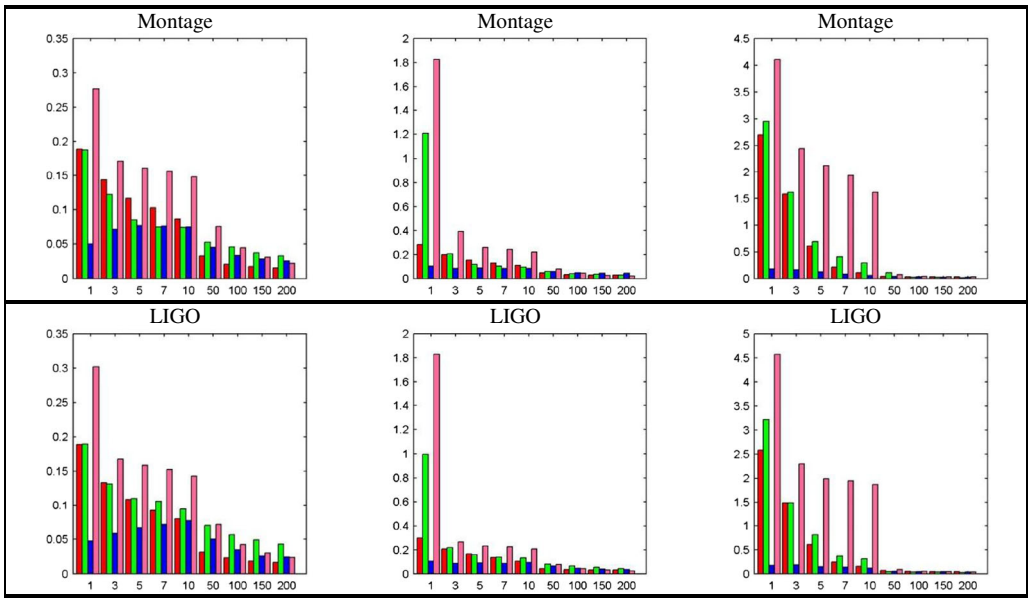
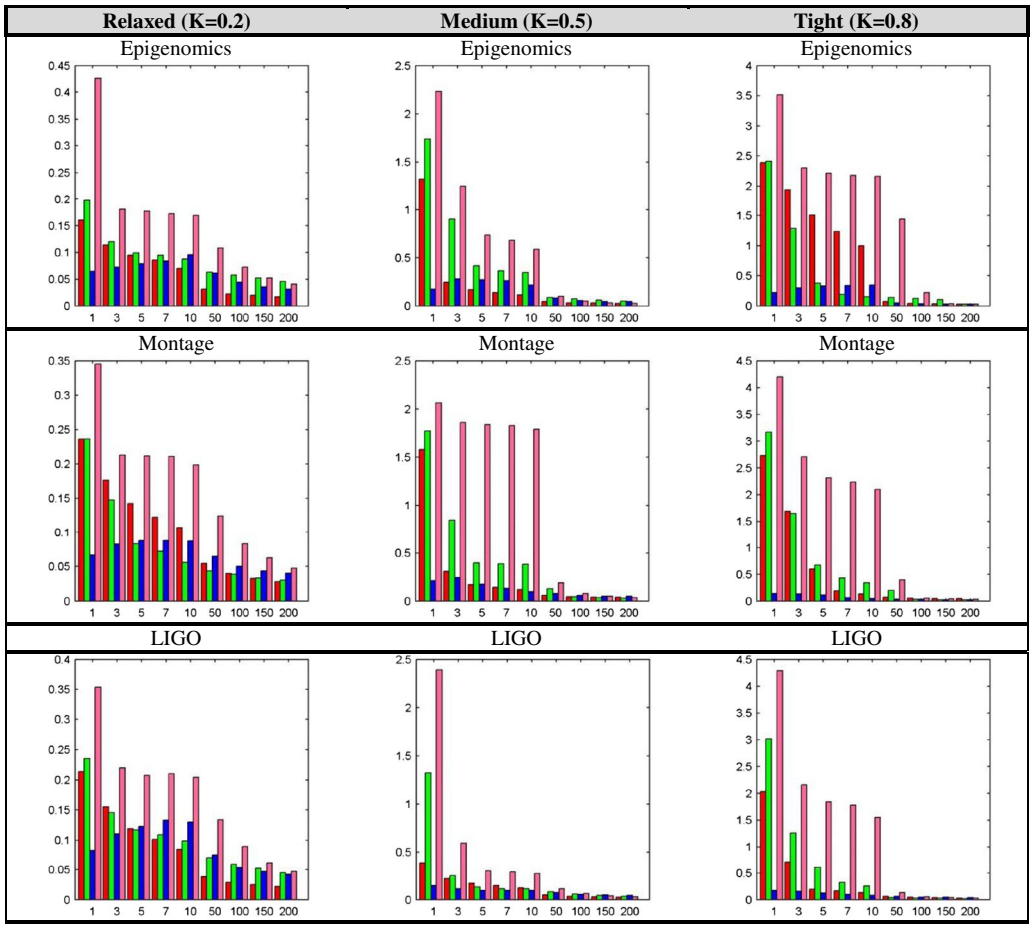


Fig. 12 The proposed stability metric for different workflows and constraints, (**a**) small workflows, (**b**) medium workflows, and (**a**) large workflows



(b) Medium Workflows



(c) Large Workflows

Fig. 12 (continued)

Table 4 Workflow execution times (a-1), (a-2), and (a-3) for 200 generations (20000 iterations for PAES); (b-1), (b-2), and (b-3) for 500 generations (50000 iterations for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	13.51	11.68	12.18	11.37	12.29	12.43	12.30	13.23	13.14
NSGA-II	15.80	15.92	17.25	14.61	14.79	16.05	18.17	20.70	21.14
PAES	71.98	66.43	70.74	55.81	58.81	54.34	59.11	67.75	66.46
SPEA2	108.1	106.1	105.4	105.8	105.5	104.7	111.0	110.7	109.0

(a-1) Small Workflow and 200 Generations (20000 for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	14.29	14.75	15.24	17.00	17.19	17.68	14.80	15.47	16.15
NSGA-II	28.59	32.81	33.13	33.93	35.05	38.07	33.37	34.81	36.68
PAES	86.60	82.45	77.34	82.40	80.17	76.22	84.16	85.03	78.22
SPEA2	134.3	133.0	132.5	142.9	143.2	143.6	138.9	137.2	141.7

(a-2) Medium Workflow and 200 Generations (20000 for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	21.94	23.18	22.62	26.78	27.05	28.09	21.99	22.39	22.56
NSGA-II	97.79	106.5	104.5	106.9	115.5	118.2	100.3	104.1	106.8
PAES	175.2	177.7	172.0	184.0	185.1	182.6	173.8	173.2	171.4
SPEA2	256.6	256.3	260.8	279.0	282.3	283.2	259.2	262.7	260.5

(a-3) Large Workflow and 200 Generations (20000 for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	29.36	29.39	30.99	29.05	30.09	31.00	31.22	32.39	33.44
NSGA-II	35.28	37.61	39.39	35.69	39.38	39.54	42.63	46.24	48.04
PAES	179.9	181.4	179.4	160.4	163.1	161.0	197.4	198.5	200.1
SPEA2	277.8	278.2	280.2	264.0	265.4	265.8	296.3	298.4	298.2

(b-1) Small Workflow and 500 Generations (50000 for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	36.26	36.74	37.28	40.58	43.67	44.40	37.17	38.04	40.88
NSGA-II	68.58	71.56	76.69	87.25	86.76	91.32	75.57	81.49	83.95
PAES	228.6	226.1	215.7	202.0	200.6	204.0	229.6	301.2	229.0
SPEA2	343.6	348.3	350.2	353.9	353.3	355.7	341.9	342.3	344.7

(b-2) Medium Workflow and 500 Generations (50000 for PAES)

	Epigenomics Relaxed	Epigenomics Medium	Epigenomics Tight	Montage Relaxed	Montage Medium	Montage Tight	LIGO Relaxed	LIGO Medium	LIGO Tight
LWSGA	53.46	54.83	55.79	66.88	67.25	69.56	54.33	55.00	57.46
NSGA-II	219.7	232.0	243.7	269.3	282.4	283.2	223.7	238.1	269.4
PAES	446.2	450.4	447.8	485.6	489.4	490.1	455.8	455.8	459.3
SPEA2	654.5	660.3	662.3	663.7	666.1	664.2	617.6	617.0	617.6

(b-3) Large Workflow and 500 Generations (50000 for PAES)

was implemented using two different mechanisms. The first is the method given in Section 3.3, denoted LWSGA*, and the second is the method in [30], denoted LWSGA$^{\#}$. The method in [30] penalizes all solutions that do not meet the budget or deadline constraints so the degree of violation is not considered. Thus useful solutions that result in feasible solutions in later generations are unlikely to survive. The

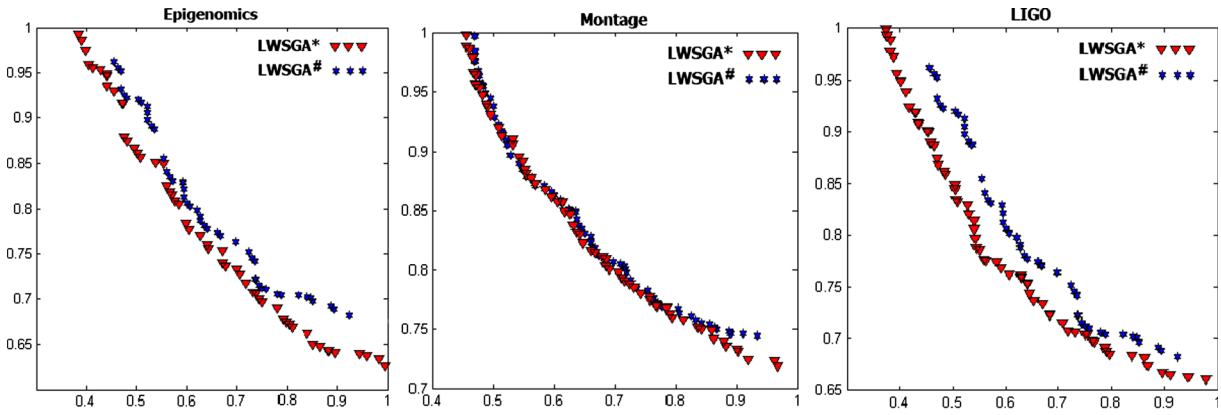


Fig. 13 The effects of penalty mechanisms on the quality and diversity of the workflow solutions

performance with the two penalty mechanisms for each of the workflows is presented in Fig. 13. Both α and ω were set to 0.1. This shows that for the Epigenomics and LIGO workflows, the LWSGA* solutions are considerably better than those with LWSGA$^{\#}$. This is because the LWSGA* solutions have better diversity in the solution space, in particular solutions near the constraint boundaries which can be important. Conversely, LWSGA$^{\#}$ has no solutions near the boundaries so the solution space is not as diverse. With the Montage workflows, the solutions obtained are very similar. In general, the proposed penalty mechanism improves algorithm performance by increasing the quality of the solutions and their diversity in the solution space.

4.5 Discussion

Results were obtained for three workflows with different structures and sizes, and four metrics were employed to evaluate the algorithms. The hypervolume indicator, I_H, confirms that the proposed method has good coverage of the solution space, while the other methods have less diverse on this space. Further, the epsilon indicator, $I_{\varepsilon+}$, and scheduling time results show that the proposed method increases the quality of the solutions, and significantly reduces the scheduling time. The convergence metric, γ, shows the potential of the proposed method to converge to non-dominated solutions, while the stability metric, $\hat{\mathcal{E}}$, indicates fast convergence to a stable solution. Overall, the proposed algorithm is more efficient and provides better solutions than the other methods.

5 Conclusion

In this paper, a new technique was presented for scheduling workflows in a grid environment based on a multi-objective genetic algorithm. This technique takes into account user requirements to create a set of solutions for workflow scheduling. Due to the inherent heterogeneity in a grid environment and the competition between users for resources, it is important to generate these solutions quickly. To achieve this goal, workflows were divided into levels and chromosomes in the genetic algorithm structured according to these levels. This eliminates the need to check task dependencies as the solutions generated will be feasible. In addition, a new fitness function was proposed to have a good diversity of solutions. The proposed method was compared with several well-known algorithms using a number of metrics. The results obtained show that this method has excellent speed and efficiency.

Future work will include developing a dynamic layering model and an adaptive fitness function to improve the performance of the algorithm. The proposed algorithm can also be adapted to support cloud computing systems.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco (1998)
2. Broberg, J., Venugopal, S., Buyya, R.: Market-oriented grids and utility computing: the state-of-the-art and future directions. J. Grid Comput. **6**(3), 255–276 (2008)

3. Deelman, E. et al.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. J. Sci. Program. **13**, 219–237 (2005)
4. Yu, J., Buyya, R.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. J. Sci. Program. **14**, 217–230 (2006)
5. Yu, J., Buyya, R.: A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. J. Sci. Program. **14**(3,4), 217–230 (2006)
6. Foster, I. et al.: The Physiology of the Grid. Open Grid Service Infrastructure WG, Global Grid Forum (2002)
7. Benkner, S. et al.: VGE - a service-oriented grid environment for on-demand supercomputing. In: Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA (2004)
8. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the askalon grid environment. In: SIGMOD Record. 34,56–62 (2005)
9. Berman, F. et al.: New grid scheduling and rescheduling methods in the grads project. J. Parallel Prog. **33**, 209–229 (2005)
10. Deelman, E. et al.: Mapping abstract complex workflows onto grid environments. J. Grid Comput. **1**, 25–39 (2003)
11. Fahringer, T. et al.: ASKALON: a tool set for cluster and Grid computing. J. Concurr. Comput. Pract. Exp. **17**, 143–169 (2005)
12. Ludäscher, B. et al.: Scientific workflow management and the KEPLER system. J. Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows (2005)
13. Neubauer, F., Hoheisel, A., Geiler, J.: Workflow-based Grid Applications. J. Futur. Gener. Comput. Syst. **22**, 6–15 (2006)
14. Oinn, T. et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17), 3045–3054 (2004). Oxford University Press, London, UK
15. Yu, J., Buyya, R.: Taxonomy of Workflow Management Systems for Grid Computing, Vol. 3. Spring Science and Business Media B.V., New York, USA (2005)
16. Wieczorek, M., Prodan, R., Hoheisel, A.: Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem. CoreGRID Technical Report Number TR-0106 (2007)
17. Wieczorek, M., Hoheisel, A., Prodan, R.: Towards a general model of the multi-criteria workflow scheduling on the grid. J. Futur. Gener. Comput. Syst. **25**, 237–256 (2009)
18. Gharooni-fard, G., Moein-darbari, F., Deldari, H., Morvaridi, A.: Scheduling of scientific workflows using a chaos-genetic algorithm. In: Proceedings Computer Science. 1, 1445–1454 (2012)
19. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
20. Haluk, T. et al.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**, 260–274 (2002)
21. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **68**(4), 399–409 (2008)
22. Daoud, M., Kharma, N.: GATS 1.0: A novel GA-based scheduling algorithm for task scheduling on heterogeneous processor nets. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 2209–2210. ACM, New York (2005)
23. Zhong, Y.W., Yang, J.G.: A genetic algorithm for tasks scheduling in parallel multiprocessor systems. In: International Conference on Machine Learning and Cybernetics, X'ian, China. 3, pp. 1785–1790 (2003)
24. Kim, S.C., Lee, S.: Push-pull: Guided search dag scheduling for heterogeneous clusters. In: International Conference on Parallel Processing, ICPP 2005, pp. 603–610. University of Oslo, Norway (2005)
25. Prodan, R., Wieczorek, M.: Negotiation-based scheduling of scientific grid workflows through advance reservations. J. Grid Comput. **8**(4), 493–510 (2010)
26. Gkoutioudi, K.Z., Helen, D.K.: Multi-criteria job scheduling in grid using an accelerated genetic algorithm. J. Grid Comput. **10**(2), 311–323 (2012)
27. Tao, Y., et al.: Dependable grid workflow scheduling based on resource availability. J. Grid Comput., 1–15 (2013)
28. Cao, F., Zhu, M.M.: Distributed workflow mapping algorithm for maximized reliability under end-to-end delay constraint. J. Supercomput., 1–27 (2013)
29. Garg, R., Singh, A.K.: Multi-objective optimization to workflow grid scheduling using reference point based evolutionary algorithm. J. Comput. Appl. **22**(6), 44–49 (2011)
30. Yu, J., Buyya, R.: Multi-objective planning for workflow execution on Grids. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 10–17 (2007)
31. Talukder, A., Kirley, M., Buyya, R.: Multiobjective Differential Evolution for Scheduling Workflow Applications on Global Grids. John Wiley, Chichester (2009)
32. Bhat, D.N.: An evolutionary measure for image matching. In: Fourteenth International Conference on Pattern Recognition, pp. 850–852. IEEE Press, New York (1998)
33. He, X., Sun, X.H., Laszewski, G.V.: QoS guided min–min heuristic for grid task scheduling. J. Comput. Sci. Technol. **18**(4), 442–451 (2003)
34. Tao, Q., Chang, H., Yi, Y., Gu, C.: A Grid Workflow Scheduling Optimization Approach for e-Business Application. In: International Conference on E-Business and E-Government, pp. 198–171 (2010)
35. Verboven, S., Hellinckx, P., Arickx, F., Broeckhove, J.: Runtime Prediction Based Grid Scheduling of Parameter Sweep Jobs: In: Proceedings Asia-Pacific Conf. Services Computing, pp. 33–38 (2008)
36. Holland, J.H.: Adaptation in natural and artificial systems. In: Ann Arbor: University of Michigan Press (1975)
37. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley, England (2001)

38. Konak, A., Coit, D.W., Smith, A.E.: Multi-objective optimization using genetic algorithms: A tutorial. J. Reliab. Eng. Syst. Saf. **91**(8), 992–1007 (2006)
39. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
40. Knowles, J.D., Corne, D.: The Pareto Archived Evolution Strategy:A New Baseline Algorithm for Pareto Multiobjective Optimisation. In: Proceedings of the Congress on Evolutionary Computation (CEC'99). 1, pp. 98–105 (1999)
41. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
42. Schaffer, J.D.: Multiple objective optimization with vector evaluated genetic algorithms. In: Proceedings of the International Conference on Genetic Algorithm and their Applications (1985)
43. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Evolutionary Methods for Design, Optimization, and Control. 1, pp. 19–26 (2002)
44. Yen, G.G., Lu, H.: Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. IEEE Trans. Evol. Comput. **7**(3), 253–74 (2003)
45. Hirales-Carbajal, A., et al.: Multiple workflow scheduling strategies with user run time estimates on a grid. J. Grid Comput. **10**(2), 325–346 (2012)
46. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Cost-driven scheduling of grid workflows using partial critical paths. IEEE Trans. Parallel Distrib. Syst. **23**(7), 1400–1414 (2012)
47. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of Scientific Workflows. In: Third Workshop Workflows in Support of Large Scale Science (2008)
48. Knowles, J., Thiele, L., Zitzler, E.: A tutorial on the performance assessment of stochastic multiobjective optimizers. Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland (2006)
49. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comput. **7**(2), 117–132 (2003)
50. Deb, K., et al.: A fast elitist multi-objective genetic algorithm: NSGA-II. J. Parallel Probl. Solv Nat. **VI**, 849–858 (2002)
51. Singh, G., Kesselman, C., Deelman, E.: Optimizing grid-based workflow execution. J. Grid Comput. **3**(3–4), 201–219 (2006)