A Knowledge Guided Multi-Population Evolutionary Algorithm for Dynamic Workflow Scheduling Problem

Jingyuan Xu*‡, Jiajian Yang*‡, Peiru Li*‡, Ziming Wang*, Changwu Huang*, and Xin Yao†

*Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation,
Department of Computer Science and Engineering,
Southern University of Science and Technology, Shenzhen 518055, China.
†Department of Computing and Decision Sciences, Lingnan University, Hong Kong SAR.

***Abstract*—The workflow scheduling problem is a fundamental task in cloud computing. This paper addresses the challenge of workflow scheduling in dynamic and uncertain cloud environments, where computing resources may become inaccessible due to hardware or software failures. To tackle this challenge, we propose a novel algorithm called the Order Feature Guided Multi-Population (OFGMP) algorithm for dynamic workflow scheduling in cloud environments. The OFGMP algorithm utilizes a multi-population evolutionary framework, incorporating a knowledge-guided reproduction operator that leverages the order feature of solutions, as well as repair mechanisms to adapt to changing environmental conditions. Extensive experiments are conducted to validate the algorithm's performance against existing dynamic scheduling approaches. The experimental results demonstrate the superiority of our proposed method over others on a number of test cases.**

***Index Terms*—Workflow Scheduling Problem, Cloud Computing, Evolutionary Algorithm, Multi-objective Optimization**

I. Introduction

With the rapid development of cloud computing, the workflow scheduling problem (WSP) is becoming increasingly important. The primary objective of WSP is to find the optimal or multiple viable scheduling schemes for assigning interdependent tasks within a workflow to various virtual machines, which are also referred to as instances and used hereafter in this paper. Due to the complexity of the workflow topology and the dynamics and heterogeneity of cloud computing resource pools, the WSP in cloud computing is widely recognized as a NP-complete problem [1].

Traditionally, the WSP is modeled as a static optimization problem, either as a single objective optimization problem, which focuses on optimizing a single objective, or a multi-objective optimization problem (MOP), which aims to provide multiple trade-off solutions among different objectives [1]. Makespan and cost are the two most common but conflicting quality-of-service (QoS) requirements. Makespan signifies the overall completion time of a workflow, while cost denotes the total expenses associated with executing the workflow.

In real-world scenarios, the WSP is inherently dynamic [2], [3] since the performance of the resource pool fluctuates continuously and occurrences of hardware or software faults lead to frequent resource inaccessibility. This dynamic nature renders solutions obtained from static scheduling algorithms infeasible. Consequently, it is more appropriate to model the WSP in cloud computing as a dynamic optimization problem.

In contrast to the static WSP [1], in the dynamic WSP, some solutions may become infeasible after changes occurs over time. Static scheduling algorithms, which do not account for resource inaccessibility, are limited in their ability to handle such situations. Their only option is to restart the scheduling process. However, this approach not only wastes computational resources but also discards valuable information from previous solutions. While there have been efforts to develop dynamic scheduling algorithms tailored to address the challenges of changing environments in WSP, the existing algorithms for dynamic WSP still lack effectiveness.

To address the limitations of current dynamic workflow scheduling algorithms and tackle dynamic WSP, we propose the Order Feature Guided Multi-Population (OFGMP) algorithm for dynamic multi-objective WSP in cloud computing. First, we introduce a similarity-based repair strategy that replaces inaccessible instances with the most similar ones based on their attribute values using the Manhattan distance. Our analysis and experiments demonstrate that this strategy outperforms other repair strategies and achieves better performance. However, the similarity-based repair strategy may lead to a loss of diversity as the repaired population becomes similar to previous ones. To mitigate this issue, we incorporate newly generated feasible individuals from another population to maintain diversity by replacing some of the infeasible individuals. Furthermore, through observation and analysis of

‡These authors contributed equally to this work.

Corresponding author: Changwu Huang (huangcw3@sustech.edu.cn).

This work was supported by the National Key R&D Program of China (Grant No. 2023YFE0106300), the National Natural Science Foundation of China (Grant No. 62250710682), the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No.2017ZT07X386).

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00014

non-dominated solutions in the Pareto optimal front (POF) for specific workflows, we discover that certain tasks are preferably executed in a specific order on the same instance, which we term "order feature" (OF). We leverage this order feature knowledge to guide the evolution of inferior solutions. The performance of our proposed OFGMP algorithm has been validated through a comprehensive experimental study.

The remainder of this paper is organized as follows. Section II describes the background of dynamic WSP, including the formulation of WSP and the related work for dynamic multi-objective WSP. Our proposed method, i.e., OFGMP algorithm, is described in Section III. Section IV presents the experimental study to validate the performance of the proposed OFGMP algorithm. Finally, the paper is concluded in Section V.

II. Background

In this section, we first describe the dynamic WSP investigated in this paper. Then, some related work is discussed.

A. Dynamic Workflow Scheduling Problem

A workflow is composed of various tasks or nodes interconnected by dependencies or data flows, which can be depicted as a directed acyclic graph, denoted as $W = (T, E)$, where $T = \{t_1, t_2, \cdots, t_n\}$ is the set of tasks and $E = \{e_{i,j}\} = \{(t_i, t_j) | t_i \in T, t_j \in T, i \neq j\}$ is a collection of edges where each edge $e_{i,j}$ denotes the data or control dependency between tasks t_i and t_j. The execution time of a task $t_i \in T$ on an instance (i.e., a virtual machine) with a single compute unit is known as the reference time, which is denoted as $RT(t_i)$. The data transfer size from task t_i to t_j is represented by $data(t_i, t_j)$ for $(t_i, t_j) \in E$. For a task t_j, its predecessors are denoted as $pred(t_j) = \{t_i | (t_i, t_j) \in E\}$, and its successors are presented by $succ(t_j) = \{t_k | (t_j, t_k) \in E\}$. In a given workflow W, the task with no predecessor is denoted as t_{entry}, and the task with no successor is denoted as t_{exit}.

A workflow is scheduled and executed in a distributed computing platform consisting of a cluster of instances (i.e., virtual machines) $R = \{r_1, r_2, \cdots, r_m\}$, where each instance $r_i \in R$ involves a set of attributes including the computing capacity (which describes the CPU capacity of the instance, i.e., the number of compute units) $CU(r_i)$, the bandwidth $B(r_i)$, and the price $P(r_i)$. Consequently, given the reference time $RT(t_i)$, the actual execution time of task t_i on instance r_k can be calculated as [3],

$$ET(t_i, r_k) = \frac{RT(t_i)}{CU(r_k)}. \tag{1}$$

The transfer time or communication time between task t_i and t_j can be computed as [3]:

$$TT(t_i, t_j) = \begin{cases} \dfrac{data(t_i, t_j)}{\min\{B(r_p), B(r_q)\}}, & p \neq q \\ 0 \qquad\qquad\quad , & p = q \end{cases} \tag{2}$$

where r_p and r_q are the instances to which t_i and t_j are scheduled, respectively.

In our study, the goal of WSP is to minimize both the makespan and cost. The makespan represents the total time required to complete a workflow, while the cost corresponds to the charge incurred from renting instances to finish a workflow using some pricing model, like Amazon EC2's on-demand billing model [4], where usage time of less than one hour is rounded up to the nearest hour. The usage time of a task t_j on an instance r_q is determined by the start time $ST(t_j, r_q)$ and finish time $FT(t_j, r_q)$, which are computed in order as [3]:

$$ST(t_{entry}, r_q) = ready(r_q), \tag{3}$$

$$ST(t_j, r_q) = \max\{ready(r_q), \max\{FT(t_i, r_p) | t_i \in pred(t_j)\}\}, \tag{4}$$

$$FT(t_j, r_q) = ST(t_j, r_q) + ET(t_j, r_q) + \sum_{t_k \in succ(t_j)} TT(t_j, t_k), \tag{5}$$

where $ready(r_q)$ represents when the instance is ready to be executed. Finally, the makespan and cost are computed as [3],

$$Makespan = \max\{FT(t_i, r_k) | t_i \in T, r_k \in R\}, \tag{6}$$

$$Cost = \sum_{r_k \in R} P(r_k) \cdot \left[\max_{t_i \in T} FT(t_i, r_k) - \min_{t_j \in T} ST(t_j, r_k) \right]. \tag{7}$$

In this paper, we take into account the dynamic nature of computing resources in distributed platforms, as real-world environments may experience instances becoming inaccessible due to hardware or software factors. To quantify such environmental changes, we characterize it by the changing frequency f and severity s. The changing frequency f denotes the rate of change occurrence, and the severity s represents the percentage of instances becoming inaccessible at each change. To be specific, given the available instances that can be allocated in the cloud platform $R = \{r_1, r_2, \cdots, r_m\}$, where m is the number of instances, when the resource inaccessibility occurs, some instances, denoted as $R_{inacc} = \{r_j | r_j \in R\}$ with $|R_{inacc}| = m \cdot s$, become inaccessible. Consequently, the accessible instances that can be scheduled for a workflow are $R_{acc} = R \setminus R_{inacc} = \{r_i | r_i \in R \text{ and } r_i \notin R_{inacc}\}$. Thus, in the dynamic WSP addressed in this work, the accessible instances R_{acc} dynamically change in the scheduling process.

The dynamic changes in the resource pool have two main impacts on cloud computing workflow scheduling. Firstly, solutions found by scheduling algorithms may become infeasible as they include inaccessible instances in the scheduling plans. Secondly, the optimality of the obtained solutions is affected by the changes in computing resources. Therefore, it is essential to address these challenges associated with the dynamic WSP.

B. Related Work

Qiu *et al.* [5] introduced DMGA, which utilizes the longest common sequence in the solution set to guide solution evolution, ensuring both rapid convergence and diverse solutions. Han *et al.* [6] proposed CMSWC, which employs search trees

and heuristics to narrow down the solution space. Belgacem *et al.* [7] combined ant colony optimization with MOHEFT [8] to effectively explore extensive solution spaces. Vavak *et al.* [9] introduced DNSGA-II-HM, which dynamically adjusts algorithm parameters to direct the search process. However, these approaches do not consider instances becoming inaccessible and therefore cannot be directly applied in dynamic WSP. Although some of them can be adapted by adding a repair strategy, the convergence of solutions is significantly affected when the environment changes.

As dynamic WSP gained more attention, several approaches for dynamic WSP in cloud environments were proposed. Ismayilov *et al.* [3] proposed NN-NSGA-II, combined neural networks with NSGA-II [1] to predict potential solutions in changing environments. Dong *et al.* [10] develop a double deep Q network framework (DDQN) based on reinforcement learning to adaptively select re-submission and replication strategy considering fault tolerance. Deb *et al.* [11] proposed DNSGA-II-B, which focuses on preserving population diversity to enhance global search capabilities and prevent premature convergence. Similarly, by maintaining solution diversity, Carlos *et al.* [12] proposed DNSGA-II-gIDG by incorporating both random and mutated immigrants into the population to ensure diversity in each generation. Rani *et al.* [13] introduced an ant-lion colony optimization method, utilizing ant-lions to capture the characteristics of ants during the search. Koo *et al.* [14] developed MB-NSGA-II, which uses a memory archive to store recent Pareto sets and retrieves them upon environmental changes. However, these algorithms lack strategies to improve solution diversity when the environment changes, leading to the risk of falling into local optima.

III. Order Feature Guided Multi-Population (OFGMP) Algorithm

In this section, we present our proposed Order Feature Guided Multi-Population (OFGMP) algorithm for dynamic WSP. The overall procedure of the algorithm is introduced first, and our proposed order feature guided reproduction (OFGR) operator and repair strategies are described.

The OFGMP algorithm utilizes a multi-population framework that incorporates distinct evolutionary and repair strategies to adapt to environmental changes (i.e., changes in computing resources). We established two distinct populations: a normal population and a feature population, which are involved in our proposed OFGR operator. The overall process of OFGMP is described in Algorithm 1.

Before describing the operators of our algorithm, we first introduce the representation of individuals in our algorithm. A scheduling scheme for a workflow usually involves two aspects, the execution order of tasks and the task-to-instance mapping. Thus, we use a similar encoding to the one used in [1], that is, the individual representation contains two sequences: 'order' and 'task2ins'. The 'order' sequence records the serial number of tasks and satisfies the topological relationship between tasks of the workflow. The 'task2ins' sequence represents the mapping between each task and the instance.

Algorithm 1: Order feature Guided Multi-Population (OFGMP) Algorithm

Input : Population size n, Maximal generation g.
Output: Non-dominated solution set $\mathcal{S}$

1 $\mathcal{P}_n \leftarrow$ Initialize a population with n/2 individuals
2 $\mathcal{P}_f \leftarrow$ Initialize a population with n/2 individuals
3 **for** $i = 1, 2, \cdots, g$ **do**
4 **while** $|\mathcal{P}_n| + |\mathcal{P}_f| < 2 \times n$ **do**
5 $p_1, p_2 \leftarrow$ Randomly select from $\mathcal{P}_n$
6 $p_3, p_4 \leftarrow$ Randomly select from $\mathcal{P}_f$
7 **if** rand(0,1) < 0.5 **then**
  `// Inter-Population Reproduce`
8 $(c_1, c_3) \leftarrow$ Generate offspring of p_1, p_3
9 $(c_2, c_4) \leftarrow$ Generate offspring of p_2, p_4
10 **else** `// Intra-Population Reproduce`
11 $(c_1, c_2) \leftarrow$ Generate offspring of p_1, p_2
12 $(c_3, c_4) \leftarrow$ OFGR$(p_3, p_4, \mathcal{P}_f)$
13 **end**
14 $\mathcal{P}_n \leftarrow \mathcal{P}_n \cup \{c_1, c_2\}$
15 $\mathcal{P}_f \leftarrow \mathcal{P}_f \cup \{c_3, c_4\}$
16 **end**
17 $\mathcal{P}_n \leftarrow$ Select best n/2 individuals from $\mathcal{P}_n$
18 $\mathcal{P}_f \leftarrow$ Select best n/2 individuals from $\mathcal{P}_f$
19 **if** change occurs **then**
  `// Repair strategies in Section III-B`
20 Perform individual substitution for $\mathcal{P}_n$
21 Perform similarity-based repair for $\mathcal{P}_f$
22 **end**
23 $\mathcal{S} \leftarrow$ Non-dominated solution set of $\mathcal{P}_n \cup \mathcal{P}_f$
24 **end**

Besides our proposed OFGR operator, the genetic operators in [1] are also used to generate offspring as in Lines 8, 9, 11 of Algorithm 1 and in Line 1 of Algorithm 2.

To begin with, the OFGMP algorithm randomly generates the initial normal population $\mathcal{P}_n$ and initial feature population $\mathcal{P}_f$, together with population size n (Lines 1-2). In this step, the 'order' of all individuals in the population is randomly generated using a random topological sorting approach. The 'task2ins' sequence is initialized differently for each individual: half of the individuals have random task-to-instance mappings, while the other half allocate all tasks to the same randomly selected instance.

The algorithm then enters the main loop. When generating individuals, two pairs of parents (p_1, p_2) and (p_3, p_4) are randomly selected from population $\mathcal{P}_n$ and $\mathcal{P}_f$, respectively (Lines 5-6). The inter-population reproduction and intra-population reproduction approaches are randomly chosen to generate new offspring (Lines 7-13) and these new individuals are added to their respective populations (Lines 14-15). Inter-population reproduction utilizes the crossover and mutation operators in [1] to create offspring. Intra-population reproduction incorporates our proposed OFGR (order feature guided

reproduction) operator, which is described in Algorithm 2.

After generating new individuals, OFGMP employs fast non-dominated sorting and fast crowding distance sorting [15] on $\mathcal{P}_n$ and $\mathcal{P}_f$, and updates them by the best n individuals after sorting (Lines 17-18). If environment change occurs, OFGMP executes repair strategies (Lines 19-22). For population $\mathcal{P}_n$, we execute an individual substitution repair strategy, which means replacing infeasible solutions with randomly generated ones. For population $\mathcal{P}_f$, OFGMP performs similarity-based repair. Details on different repair strategies are provided in Section III-B.

A. Order Feature Guided Reproduction Operator

The individual or chromosome used in this work includes two parts, that is, the 'order' and 'task2ins' sequences, where the 'order' sequence is a permutation of all tasks in a workflow, and the 'task2ins' represents the assignment of tasks to instances. The order of task execution (task order) is crucial, especially when tasks are executed in the same instance. Take the workflow illustrated in Fig. 1 as an example, where tasks 1, 2, and 3 represent completed tasks, and tasks 4, 5, and 6 signify pending tasks. We notice that when tasks 4, 5, and 6 are on separate instances, their execution order is less important. However, if they are on the same instance, certain task order sequences like 4→5→6 and 5→4→6 are better than task orders like 6→4→5 or 6→5→4, because they allow subsequent tasks 7 and 8 to execute earlier, optimizing overall performance. This indicates that, in a solution, if multiple tasks are assigned to an instance, the execution order of these tasks makes a difference to the quality of the schedule, though the task-to-instance sequence remains the same. In other words, keeping the 'task2ins' sequence the same, the 'order' with specific features could be better than others. This finding has motivated us to exploit the order feature (i.e., the feature of the order of multiple tasks on an instance) to generate good-quality solutions. Hence, we propose an order feature guided reproduction (OFGR) operator, which is described below.

The basic idea of OFGR operator is to exploit the knowledge involved in the orders of non-dominant solutions found so far to guide the generation of new individuals. In OFGP operator, we first generate individuals according to the crossover and mutation operators in NSGA-II [1], and then the order feature is utilized to further improve the generated individuals. The procedure of OFGR operator is described in Algorithm 2.

Firstly, two offspring individuals c_3 and c_4 are generated based on the two parents p_3 and p_4. Then the objective space of population $\mathcal{P}_f$ is divided into five subspaces (regions), denoted as $\{\Delta_i\}_{i=1}^{5}$, and each individual in $\mathcal{P}_f \cup \{c_3, c_4\}$ is assigned to one of these subspaces. The division of the objective space and the assignment of individuals to subspaces follow the approach used in [16], as illustrated in Fig. 2. This division allows us to group dominating solutions that are close in the objective space, thus providing stronger guidance for generating new individuals in each region. Hence, by extracting order features of dominating solutions in a region, we can use these features to improve the quality of generated solutions in this region.

If the parent individuals p_3 and p_4 belong to the same subspace Δ_k, the OFGR operator will improve their offspring individuals c_3 and c_4 using the order feature (Lines 3-32). For each individual $s \in \mathcal{S}$, where $\mathcal{S}$ represents the set of non-dominated solutions associated with subspace Δ_k from $\mathcal{P}_f$, we collect the order features of instances that have been assigned multiple tasks into a set $\mathcal{F}$ (Lines 9-14). To reduce the size, we only save partial orders whose length is between 1 and 1/4 of the total number of tasks in set $\mathcal{Q}$ (Lines 15-19).

Subsequently, one order feature F is randomly selected from the saved order feature set $\mathcal{Q}$, and the adjacent task pairs in F, that is, $(F[j-1], F[j])$ for $j = 1, 2, \cdots, |F|-1$, are added as new edges to a copy of the workflow's Directed Acyclic Graph (DAG), denoted as G (Line 21-24). The order of the offspring individual c_3 and c_4 are updated with randomly topological sorting on G (Lines 25-26). Finally, to ensure that all the tasks in order feature are executed in the same instance, we randomly select two instances r_1, r_2 from accessible instances set and assign every task in order feature of c_3, c_4 to r_1, r_2, respectively (Lines 27-31).

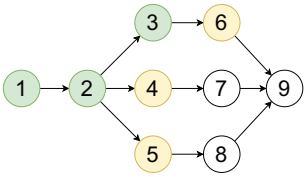


Fig. 1. An example of workflow with 9 tasks.

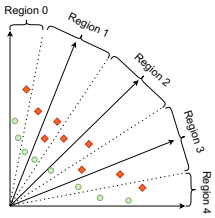


Fig. 2. Regions and Individuals in the Feature Population.

B. Repair Strategies

When the environment changes, solutions can become infeasible and require repairs for adaptation. One common approach for such repairs is the random repair strategy [3], where tasks on the inaccessible instance are redistributed randomly to another accessible one [2], as shown in Fig. 3(a). In this work, we propose another two repair strategies: similarity-based repair strategy and individual substitution strategy.

The similarity-based repair strategy, as shown in Fig. 3(b), aims to generate solutions that are closer to post-convergence but may reduce population diversity. This strategy evaluates instance attributes, including computing unit, bandwidth, and price, using Max-Min normalization and Manhattan distance to measure similarity. When an instance becomes inaccessible, the strategy selects the instance with the highest similarity to substitute it.

The individual substitution strategy (as shown in Fig. 3(c)) focuses on enhancing population diversity. It replaces affected individuals with randomly generated ones, thereby introducing new genetic material into the population.

By combining these two repair strategies, we can improve both the diversity and convergence of the population. The effectiveness of this combination is reported in Section IV.

Algorithm 2: OFGR Operator

Input : Parent individual p_3 and p_4, feature population $\mathcal{P}_f$
Output: Offspring individual c_3 and c_4

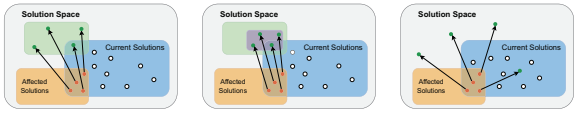
$(c_3, c_4) \leftarrow$ Generate two offspring based on p_3 and p_4
$\{\Delta_i\}_{i=1}^{5} \leftarrow$ Divide the objective space of $\mathcal{P}_f$ into five subspaces
if p_1 and p_2 belong to same subspace Δ_k **then**
  $G \leftarrow$ copy the DAG of the workflow
  $n \leftarrow$ number of tasks in the workflow
  $\mathcal{R} \leftarrow$ Solutions in $\mathcal{P}_f$ within the subspace Δ_k
  $\mathcal{S} \leftarrow$ Select non-dominated solutions from $\mathcal{R}$
  $\mathcal{Q} \leftarrow \emptyset$
  foreach $s \in \mathcal{S}$ **do**
    $\mathcal{F} \leftarrow$ Initialize an empty dictionary of lists with keys of instances in s and values of empty lists
    `// Extract order features`
    foreach $t \in s$.order **do**
      $r \leftarrow s$.task2ins[t]
      Append t to $\mathcal{F}[r]$
    end
    `// Select order features`
    foreach Order sequence $\mathcal{F}[r] \in \mathcal{F}$ **do**
      if $|\mathcal{F}[r]|>1$ and $|\mathcal{F}[r]|<n/4$ **then**
        Add the task order $\mathcal{F}[r]$ to $\mathcal{Q}$
      end
    end
  end
  $F \leftarrow$ Random select one order feature from $\mathcal{Q}$
  for $j = 1, 2, \cdots, |F| - 1$ **do**
    Add edge $(F[j-1], F[j])$ to G
  end
  `// Adjust` c_3 `and` c_4 `to satisfy the order feature`
  c_3.order $\leftarrow$ Random topological sorting of G
  c_4.order $\leftarrow$ Random topological sorting of G
  $r_1, r_2 \leftarrow$ random select two accessible instances
  foreach $t \in F$ **do**
    $c3$.task2ins[t]$\leftarrow r_1$
    $c4$.task2ins[t]$\leftarrow r_2$
  end
end

IV. Experimental Study

In this section, we first provide an overview of the experimental setup. Then, we validate the proposed repair strategies discussed in Section III-B. Finally, we compare the OFGMP algorithm with other dynamic scheduling approaches and analyze the results obtained from these experiments.

A. Experimental Setting

1) Workflow Models: We examined five types of real-world scientific workflows: Montage (I/O intensive), CyberShake (data intensive), Epigenomics, Sipht (CPU intensive), and Inspiral (LIGO), as published in a workflow project [17]. These workflows incorporate diverse structural elements such as pipelines, data distribution/aggregation, and redistribution, and thus are commonly used for assessing the scheduling algorithm's performance. Table I lists their key characteristics.



(a) Randomly Repair (b) Similarity-based Repair (c) Individual Substitution

Fig. 3. Illustration of three different repair strategies.

TABLE I
Scientific workflows and their characteristics. $|T|$ and $|E|$ refer to the number of tasks and edges.

Workflow	$\|T\|$	$\|E\|$	Avg. edge data size (MB)	Avg. task runtime (s)
Montage	100	433	3.23	10.58
CyberShake	100	380	849.60	31.53
Epigenomics	100	322	395.10	3954.90
Sipht	100	335	6.27	194.48
Inspiral	100	319	8.93	206.12

2) IaaS Service Model: Our simulation platform replicates a data center with eight EC2-based instance types [4], each offering limited capacity (seen in Table II). The simulation uses a 60-minute interval on-demand billing model. To mimic a dynamic environment, we simulated environmental changes at specific intervals. For evolutionary algorithms spanning 500 generations, low-frequency environmental change occurs at the 250th generation, and high-frequency changes occur in every 100 generations. For list-based scheduling with 100 tasks, low-frequency change occurs at the 50th task, and high-frequency changes occur in every 20 tasks. The severity of change is set at 0.03, 0.08, and 0.15 [2]. This model aims for practical relevance and allows for repeatable evaluations without the constraints of actual deployment.

3) Performance Metric: In this comparative study, we utilize the hypervolume (HV) [18] indicator as the performance metric. The HV is commonly used to evaluate the performance of algorithms on multi-objective optimization problems. It measures the volume of the objective space dominated by a solution set relative to a specific reference point. A higher HV value indicates better convergence towards the true Pareto front and increased diversity along this frontier.

To calculate the HV, the objective values of the scheduling solutions are first normalized using the upper bounds obtained in a specific dynamic case for all the experimental algorithms. The reference point for each objective is set to 1.1, as suggested in literature [19]. The HV is computed independently for each of the 20 runs, and the reported value

represents the mean HV across these runs. By employing the HV indicator, we can assess and compare the performance of different algorithms in solving dynamic WSP.

TABLE II
INSTANCE TYPES AND THEIR SPECIFICATIONS.

Type	Compute Unit	Bandwidth	Price ($)	Capacity
1	1.7/8	39,321,600	0.06	10
2	3.75/8	85,196,800	0.12	10
3	3.75/8	85,196,800	0.113	10
4	7.5/8	85,196,800	0.24	10
5	7.5/8	85,196,800	0.225	10
6	15/8	131,072,000	0.48	10
7	15/8	131,072,000	0.45	10
8	30/8	131,072,000	0.9	10

4) Comparative Algorithms: In our study, we compared the performance of OFGMP with several algorithms, including DNSGA-II-B [11], DNSGA-II-gIDG [12], MB-NSGA-II [14], DMGA [5] and CMSWC [6].To ensure a fair comparison, the configuration of these algorithms was carefully selected based on the optimal settings recommended in their respective literature. Specifically, for the evolutionary algorithms—DNSGA-II-B, DNSGA-II-gIDG, MB-NSGA-II, DMGA, and OFGMP—we standardized the population size at 200 and set the mutation rate at 0.01, aligning with the best practices for these methods. Conversely, for CMSWC, which follows a heuristic approach, we configured the algorithm with a solution count of 50 and an exploitation rate of 0.3.

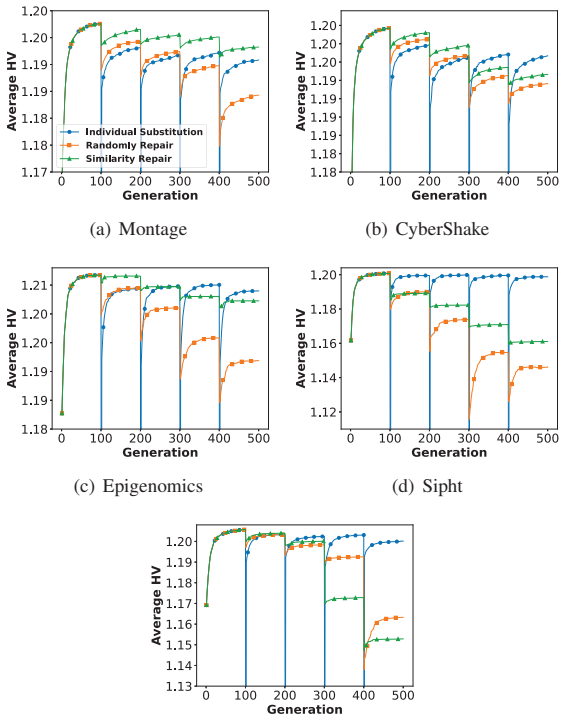
B. Results and Analysis

1) Validation of the Effectiveness of Proposed Repair Strategies: To validate the effectiveness of the proposed repair strategies, we conducted experiments using NSGA-II as the base algorithm under a severe dynamic environment case (f=100, s=0.15). We compared three repair strategies: individual substitution, random repair, and similarity-based repair. The experiments were independently performed 20 times using five different workflow models.

The mean HV values were used to evaluate the performance of each repair strategy. Fig. 4 illustrates the evolution of the mean HV values. As observed, when the environment changes, all repair strategies experience a decrease in HV values to varying degrees. However, the similarity-based repair strategy exhibited the smallest drop in HV value, indicating its effectiveness in maintaining population convergence.

In most cases, the final HV value achieved by the individual substitution strategy was the highest. This suggests that it contributes to increasing population diversity and escaping local optima after environmental changes occur. Overall, these results validate the positive impact of the proposed repair strategies on the algorithm's performance in handling environmental changes.

2) Comparative Results: We compared the performance of OFGMP with five state-of-the-art algorithms using the HV metric, and the results are presented in Table III. Considering the average HV values of each algorithm, our proposed



(a) Montage (b) CyberShake

(c) Epigenomics (d) Sipht

(e) Inspiral

Fig. 4. Comparison of 3 repair strategies on five workflows.

OFGMP achieved the largest values in 25 out of 30 test cases and consequently gains the best average ranking. Furthermore, statistical tests confirmed that OFGMP significantly outperformed the compared algorithms in the majority of the test cases. These findings demonstrate that OFGMP is an effective method for addressing dynamic WSP.

In particular, OFGMP ranked first in all test cases for the Epigenomics, Inspiral, and Montage workflows. For the CyberShake workflow, OFGMP performed the best in all cases except for the severity 0.03 and frequency 250 scenario, where it ranked third. However, statistical analysis revealed no significant differences among the algorithms in this particular test case, making them statistically equivalent.

It is worth noting that in the Epigenomics test cases, OFGMP consistently outperformed other algorithms due to the specific structure of the Epigenomics workflow. Many tasks in this workflow have only one predecessor and one successor, indicating that executing all tasks in a single chain on the same instance can lead to better makespan and cost by reducing transmission time. The order feature utilized in OFGMP focuses on tasks executed on the same instance, which helps restore the excellent characteristics of superior solutions. By extracting order features from superior solutions to guide inferior ones, OFGMP improves convergence in the Epigenomics test cases.

For the Sipht workflow, OFGMP ranked first in two cases

TABLE III

RESULTS OF OFGMP AND OTHER ALGORITHM ON AVERAGE HV. THE NUMBER IN PARENTHESES INDICATES THE RANKING OF THE ALGORITHM IN THAT TEST CASE. SYMBOLS "+","-", AND "≈" ARE USED TO INDICATE STATISTICALLY SIGNIFICANT SUPERIORITY, EQUIVALENCE, OR INFERIORITY OF OFGMP COMPARED TO EACH ALGORITHM RESPECTIVELY, BASED ON STUDENTS' T TEST (20 REPETITIONS). THE BOLDED NUMBERS IN EACH ROW REPRESENT THE ALGORITHM THAT ACHIEVED THE BEST RANKING ON THE CORRESPONDING TEST CASE WITH THREE DECIMAL PLACES RETAINED.

Workflow	Severity	Frequency	OFGMP	DMGA	DNSGAIIB	DNSGAIIgIDG	MBDNSGAII	CMSWC
CyberShake	0.03	100	**1.18883(1)**	1.18589(5+)	1.18804(3≈)	1.18499(6+)	1.18855(2≈)	1.18690(4≈)
		250	1.18823(3)	1.18475(5≈)	**1.18838(1≈)**	1.18456(6≈)	1.18836(2≈)	1.18674(4≈)
	0.08	100	**1.18793(1)**	1.18171(6+)	1.18605(3≈)	1.18425(4+)	1.18671(2≈)	1.18344(5+)
		250	**1.18790(1)**	1.18266(6+)	1.18723(4≈)	1.18430(5+)	1.18732(3≈)	1.18785(2≈)
	0.15	100	**1.18653(1)**	1.16653(5+)	1.16138(6+)	1.18246(2+)	1.17430(4+)	1.18160(3+)
		250	**1.19010(1)**	1.18470(6+)	1.18648(5+)	1.18664(3+)	1.18652(4≈)	1.18858(2≈)
Epigenomics	0.03	100	**0.95535(1)**	0.62355(4+)	0.64423(3+)	0.56220(5+)	0.65064(2+)	0.54604(6+)
		250	**0.98483(1)**	0.57061(4+)	0.58683(2+)	0.50186(6+)	0.58637(3+)	0.50776(5+)
	0.08	100	**1.01088(1)**	0.60628(5+)	0.66541(2+)	0.64810(4+)	0.65654(3+)	0.54776(6+)
		250	**0.96368(1)**	0.56135(4+)	0.58109(2+)	0.52083(6+)	0.57916(3+)	0.52757(5+)
	0.15	100	**1.02278(1)**	0.54668(3+)	0.53690(4+)	0.61613(2+)	0.52739(5+)	0.50090(6+)
		250	**0.97791(1)**	0.56454(4+)	0.58193(3+)	0.55396(5+)	0.58527(2+)	0.53637(6+)
Inspiral	0.03	100	**1.15999(1)**	1.10918(6≈)	1.13905(2≈)	1.11283(5+)	1.13884(3≈)	1.13255(4≈)
		250	**1.17004(1)**	1.15199(4≈)	1.15517(3≈)	1.13426(6+)	1.15519(2≈)	1.14830(5≈)
	0.08	100	**1.15932(1)**	1.03165(5+)	1.09446(4+)	1.11780(2+)	1.02591(6+)	1.10911(3+)
		250	**1.16839(1)**	1.14764(2≈)	1.12095(6≈)	1.13019(4+)	1.12492(5≈)	1.14400(3≈)
	0.15	100	**1.12922(1)**	0.96377(5+)	0.95150(6+)	1.08344(2≈)	0.98620(4+)	1.03992(3+)
		250	**1.17496(1)**	1.14678(3+)	1.12619(5+)	1.13811(4+)	1.11401(6+)	1.15152(2+)
Montage	0.03	100	**1.16080(1)**	1.15029(5+)	1.15648(3+)	1.14802(6+)	1.15806(2≈)	1.15644(4+)
		250	**1.16092(1)**	1.15337(5+)	1.15894(3≈)	1.14975(6+)	1.15908(2≈)	1.15831(4≈)
	0.08	100	**1.16157(1)**	1.14077(6+)	1.15390(2+)	1.14861(5+)	1.15009(4+)	1.15111(3+)
		250	**1.16085(1)**	1.15169(5+)	1.15768(3≈)	1.15018(6+)	1.15701(4≈)	1.15840(2≈)
	0.15	100	**1.15755(1)**	1.12599(4+)	1.10812(5+)	1.14396(2+)	1.10788(6+)	1.14353(3+)
		250	**1.16055(1)**	1.14945(6+)	1.15378(3+)	1.15049(5+)	1.15352(4+)	1.15782(2≈)
Sipht	0.03	100	**1.12287(1)**	1.09377(4≈)	1.09501(3≈)	1.08050(5≈)	1.06564(6≈)	1.12284(2≈)
		250	1.11959(2)	1.10514(3≈)	1.10430(5≈)	1.08703(6≈)	1.10444(4≈)	**1.12930(1≈)**
	0.08	100	1.12556(2)	1.10278(3≈)	1.06267(5≈)	1.09407(4≈)	1.04275(6≈)	**1.13385(1≈)**
		250	1.11931(2)	1.10888(3≈)	1.10635(4≈)	1.09108(6≈)	1.10557(5≈)	**1.13647(1≈)**
	0.15	100	**1.14478(1)**	0.99011(4+)	0.97658(5+)	1.11407(3+)	0.89822(6+)	1.14422(2≈)
		250	1.13059(2)	1.11908(3≈)	1.11069(4≈)	1.10360(6≈)	1.10921(5≈)	**1.14482(1≈)**
Average Ranking			1.200	4.433	3.633	4.566	3.833	3.333
# Best			25	0	1	0	0	4
# +			-	21	16	23	14	14
# -			-	0	0	0	0	0
# ≈			-	9	14	7	16	16

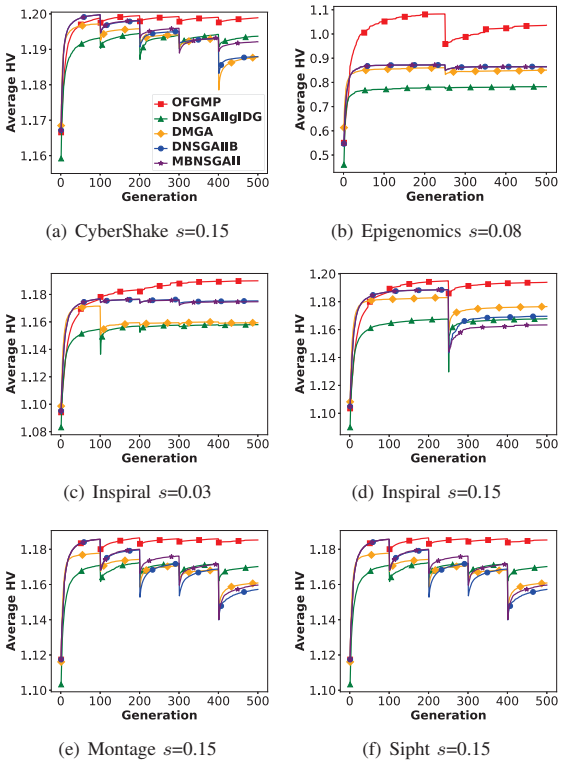
and second in the remaining three, with CMSWC performing the best. Although OFGMP and CMSWC were statistically equivalent, the suboptimal performance of our algorithm may be attributed to the presence of overlapping individuals, which does not contribute to the growth of HV.

Overall, the results in Table III indicate that OFGMP consistently outperforms other algorithms in terms of average HV. Notably, OFGMP achieved the highest number of best outcomes (#Best), surpassing CMSWC and DNSGAIIB. Across all test cases, OFGMP was either statistically superior or comparable to the competing algorithms.

Fig. 5 presents the evolution of average HV values over generations for some test cases. It is worth noting that the figures do not include the HV evolution process for CMSWC since it is not based on an evolutionary framework. From Fig. 5, it is evident that our algorithm consistently outperforms the other five algorithms on all test cases after several resource failures. This demonstrates the strong capability of our algorithm in solving dynamic multi-objective workflow scheduling problems.

V. CONCLUSION

In this work, we have presented a novel approach, called the Order Feature Guided Multi-Population (OFGMP) algorithm, to address the dynamic workflow scheduling problem in scenarios involving resource inaccessibility resulting from hardware or software issues. Building upon a multi-population evolutionary framework, our OFGMP algorithm incorporates a knowledge-guided reproduction operator that leverages the order feature of solutions, and repair mechanisms to adapt to changing environmental conditions. To evaluate the effectiveness of the proposed OFGMP algorithm, we conducted extensive experiments and compared it with several



(a) CyberShake s=0.15 (b) Epigenomics s=0.08

(c) Inspiral s=0.03 (d) Inspiral s=0.15

(e) Montage s=0.15 (f) Sipht s=0.15

Fig. 5. Average HV comparison of five algorithms over generations on six test cases (20 repetitions).

existing approaches. The experimental results demonstrate that OFGMP outperforms the five comparative algorithms significantly in the majority of the test cases. Overall, our study showcases the effectiveness of leveraging order features, implementing a similarity-based repair strategy, and utilizing a multi-population approach in addressing the challenges of dynamic workflow scheduling problems.

In the future, two possible research directions could be pursued: (1) to expand the idea of knowledge-guided operators to knowledge-guided evolutionary algorithms in general [20], [21]; and (2) to tackle dynamic scheduling problems in software engineering [22]–[24].

REFERENCES

[1] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

[2] G. Ismayilov and H. R. Topcuoglu, "Dynamic multi-objective workflow scheduling for cloud computing based on evolutionary algorithms," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 103–108.

[3] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing," *Future Generation Computer Systems*, vol. 102, pp. 307–322, 2020.

[4] "Amazon EC2 pricing," https://aws.amazon.com/cn/ec2/pricing/, 2023, accessed: 2023-11-27.

[5] H. Qiu, X. Xia, Y. Li, and X. Deng, "A dynamic multipopulation genetic algorithm for multiobjective workflow scheduling based on the longest common sequence," *Swarm and Evolutionary Computation*, vol. 78, p. 101291, 2023.

[6] P. Han, C. Du, J. Chen, F. Ling, and X. Du, "Cost and makespan scheduling of workflows in clouds using list multiobjective optimization technique," *Journal of Systems Architecture*, vol. 112, p. 101837, 2021.

[7] A. Belgacem and K. Beghdad-Bey, "Multi-objective workflow scheduling in cloud computing: trade-off between makespan and cost," *Cluster Computing*, vol. 25, no. 1, pp. 579–595, 2022.

[8] J. J. Durillo, H. M. Fard, and R. Prodan, "MOHEFT: A multi-objective list-based method for workflow scheduling," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, 2012, pp. 185–192.

[9] F. Vavak, "Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search," in *7th Int. Conf. on Genetic Algorithm*, 1997, pp. 719–726.

[10] T. Dong, F. Xue, H. Tang, and C. Xiao, "Deep reinforcement learning for fault-tolerant workflow scheduling in cloud environment," *Applied Intelligence*, vol. 53, no. 9, pp. 9916–9932, 2023.

[11] K. Deb, U. B. Rao N, and S. Karthik, "Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling," in *International Conference on Evolutionary Multi-criterion Optimization*. Springer, 2007, pp. 803–817.

[12] C. R. Azevedo and A. F. Araújo, "Generalized immigration schemes for dynamic evolutionary multiobjective optimization," in *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE, 2011, pp. 2033–2040.

[13] R. Rani and R. Garg, "Pareto based ant lion optimizer for energy efficient scheduling in cloud environment," *Applied Soft Computing*, vol. 113, p. 107943, 2021.

[14] W. T. Koo, C. K. Goh, and K. C. Tan, "A predictive gradient strategy for multiobjective evolutionary algorithms in a fast changing environment," *Memetic Computing*, vol. 2, pp. 87–110, 2010.

[15] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[16] R. Chen, K. Li, and X. Yao, "Dynamic multiobjectives optimization with a changing number of objectives," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 157–171, 2018.

[17] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, 2013.

[18] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.

[19] H. Ishibuchi, Y. Hitotsuyanagi, N. Tsukamoto, and Y. Nojima, "Many-objective test problems to visually examine the behavior of multiobjective evolution in a decision space," in *Parallel Problem Solving from Nature, PPSN XI*, R. Schaefer, C. Cotta, J. Kołodziej, and G. Rudolph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 91–100.

[20] F. Peng, K. Tang, G. Chen, and X. Yao, "Multi-start jade with knowledge transfer for numerical optimization," in *2009 IEEE congress on evolutionary computation*. IEEE, 2009, pp. 1889–1895.

[21] J. He, X. Yao, and J. Li, "A comparative study of three evolutionary algorithms incorporating different amounts of domain knowledge for node covering problem," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 2, pp. 266–271, 2005.

[22] X. Shen, L. L. Minku, R. Bahsoon, and X. Yao, "Dynamic software project scheduling through a proactive-rescheduling method," *IEEE Transactions on Software Engineering*, vol. 42, no. 7, pp. 658–686, 2015.

[23] L. L. Minku, D. Sudholt, and X. Yao, "Improved evolutionary algorithm design for the project scheduling problem based on runtime analysis," *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 83–102, 2013.

[24] X. Wu, P. Consoli, L. Minku, G. Ochoa, and X. Yao, "An evolutionary hyper-heuristic for the software project scheduling problem," in *Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings 14*. Springer, 2016, pp. 37–47.