Scientia Iranica D (2022) 29(5), 2375–2393

Sharif University of Technology
**Scientia Iranica**
*Transactions D: Computer Science & Engineering and Electrical Engineering*
http://scientiairanica.sharif.edu

# A novel discrete grey wolf optimizer for scientific workflow scheduling in heterogeneous cloud computing platforms

**M.S. Hosseini Shirvani***

*Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran.*

Received 1 December 2020; received in revised form 27 December 2021; accepted 23 May 2022

**KEYWORDS**
Cloud computing;
Scientific workflow scheduling;
Meta-heuristic algorithm;
Discrete grey wolf optimization;
Walking around technique.

**Abstract.** There are several scientific workflow applications which need a vast amount of processing. Therefore, cloud offerings are made to give them a sense of economy. Workflow scheduling has drastic impact on gaining the desired Quality of Service (QoS). The main objective of workflow scheduling is to minimize the makespan. This scheduling is formulated into a discrete optimization problem, which is NP-hard. This paper presents a novel Discrete Grey Wolf Optimizer (D-GWO) for scientific workflow scheduling problems in heterogeneous cloud computing platforms with the aim of minimizing makespan. Although traditional Grey Wolf Optimizer (GWO) has great achievements with continuous optimization problems, a clear gap exists in utilizing GWO for combinatorial discrete optimization problems given that the continuous changes in search space during the course of discrete optimization lead to inefficient or meaningless solutions. To this end, the proposed algorithm is customized to optimize the discrete workflow scheduling problem by leveraging some new binary operators and *Walking Around* approaches to balancing between exploration and exploitation in a discrete search space. Scientific unstructured workflows were investigated in different circumstances to prove the effectiveness of the proposed D-GWO. The simulation results witnessed the superiority of the proposed D-GWO to other state-of-the-arts in terms of scheduling assessment metrics.



## 1. Introduction

Cloud computing presents itself as utility computing to its users via internet protocols. It delivers unlimited heterogeneous virtual processors to solve complicated jobs [1–3]. This kind of parallel heterogeneous platform is well-suited for the execution of scientific workflows which academics are struggling with. Graph theory is used to model workflow executions. Each workflow is modeled to a Directed Acyclic Graph (DAG) in which the nodes are used for tasks and the arcs are used for data dependencies between tasks [4]. Since Virtual Machines (VMs) have different configurations, utilizing different VMs may lead to variable performance. Thus, exploiting different workflow scheduling approaches leads to different outcomes. The workflow scheduling, which determines what task should be assigned to what VM for execution, is NP-hard [5,6]. It is impossible to find optimal scheduling in a bounded time frame. To address the issue, many heuristic and meta-heuristic algorithms have been published. In this domain, the

*. *E-mail addresses: mirsaeid_hosseini@iau.ac.ir and mirsaeid_hosseini@yahoo.com*

doi: 10.24200/sci.2022.57262.5144

most important Quality of Service (QoS) parameter is *makespan* or total execution time [7,8]. Therefore, the main concentration of the proposed scheduling models is on *makespan* improvement [9]. In the scheduling context, one of the earliest algorithms is known as the list schedulers [10]. Each list scheduler firstly produces a list including ordered tasks associated with a given workflow. This list must guarantee that a topological sort of tasks does not violate the precedence constraints. It secondly assigns the high priority unscheduled task to the available VM that delivers the Earliest Finish Time (EFT). The Heterogeneous Earliest Finish Time (HEFT) algorithm is a list scheduler that exploits different ranking procedures in its first step [10]. Some extensions of the HEFT have been offered, including the Predictable Earliest Finish Time (PEFT) [11], Robust Heterogeneous Earliest Finish Time (RHEFT) [12], and Constrained Earliest Finish Time (CEFT) [13]. Moreover, variety heuristics have been proposed to enhance the results of list schedulers. The heuristics are clustering, task duplication, and data replication techniques [14–16]. In the task duplication method, few candidate tasks are duplicated to be run on some processors to increase parallelism. In the task clustering technique, some tasks are put in a cluster to be executed on the same VM [14]. By utilizing this, the data transfer time is omitted, which can potentially decrease *makespan*. At last, the data replication method uses data pipeline to reduce idle time of the VMs [16]. In larger problems, the majority of the search space remains unexplored by utilizing either heuristics or list schedulers. Therefore, miscellaneous meta-heuristics have been proposed to improve the scheduling quality, which are mainly based on the Genetic Algorithm (GA) [5,8,17,18], Particle Swarm Optimization (PSO) [1,19–21], Cuckoo Search Optimization (CSO) [22,23], and Simulated Annealing (SA) [24–28]. One of the most successful meta-heuristic algorithms, which solves continuous optimization problems, is Grey Wolf Optimization (GWO) [29,30]. The traditional GWO cannot efficiently solve discrete problems [31]. The reason is that continuous changes and modifications in the search space during the course of discrete optimization lead to inefficient or meaningless solutions. The majority of meta-heuristics have a universal trend in their exploration phase wherein they neglect exploitation of the current solution or balance between them. In this paper, a novel discrete GWO is presented to solve the combinatorial workflow scheduling problem in cloud environment with a heterogeneous platform. To this end, new binary operators and *Walking Around* techniques with a number of procedures are proposed to make a balance between exploration and exploitation of the search space.

The main contributions of this paper are as follows:

1. To present a novel Discrete Grey Wolf Optimization (D-GWO) scheduler for workflow execution, which makes a good balance between exploration and exploitation in the discrete search space;
2. To present new binary vectors and operators for both exploration (encircling the prey) and exploitation (*Walking Around* solution) for hunting and the attack process.

The rest of the paper is organized as follows. Section 2 presents related works. A review of the original GWO concepts is brought in Section 3. Section 4 provides the proposed models and problem formulation. Section 5 brings an illustrative example. Section 6 is dedicated to the proposed novel D-GWO for the workflow scheduling problem. Performance evaluation of the proposed model is given in Section 7. Section 8 includes conclusion and future direction.

## 2. Related works

Review of the scheduling algorithms helps categorize the proposed models in three classes: list-based schedulers, heuristics, and meta-heuristics. A typical list scheduler algorithm works in two steps. Firstly, it provides a valid ordered tasks list. Secondly, it assigns the high priority task to a VM that returns the EFT. The famous HEFT utilizes three ranking procedures each of which provides its own ordered list [10]. Another list scheduler is PEFT [11]. The PEFT provides a ranking algorithm according to the prediction cost table. The RHEFT [12] and Distributed HEFT (DHEFT) [13] have been proposed which take different QoS criteria [32]. The cost-effective fault tolerant workflow scheduling was suggested for the execution of real-time applications to cloud datacenter, which encounters transient and permanent failures [33]. Variety heuristics are added to improve the performance of workflow schedulers. Duplication and clustering methods are two important approaches in the area [34–36]. Task duplication copies critical tasks of a given workflow on some VMs to increase the degree of parallelism. On the other hand, the clustering method groups some highly dependent tasks in a cluster and then, all tasks in the cluster are assigned to the same VM. It potentially reduces the overall *makespan* by shortening data transfer time. However, both list schedulers and heuristic auxiliary methods cannot take over large-scale problems. They are well-suited to small-scale problems or limited time windows for quick decision. Therefore, several meta-heuristics have been extended to solve workflow scheduling problems. A shuffle-based GA was customized to solve workflow scheduling in distributed systems [8]. Another algorithm applied multi-queue besides GA to produce semi-random initialization [37]. A scheduling algorithm incorporating PSO was pro-

**Table 1.** The summary of the literature.

| Author(s)/Ref. | Classification | Approach | Advantages | Shortcomings |
|---|---|---|---|---|
| Topcuoglu et al. [10] | List-based scheduling algorithm | HEFT | It quickly ranks tasks and provisions in a list that preserves precedent constraints. | In large-scale problems, it ignores other promising possible solutions in the search space. |
| Arabnejad and Barbosa [11] | List-based scheduling algorithm | PEFT | Similar to HEFT, it provides a topological sorting list of tasks based on cost table prediction. | It does not take VM availability in the course of scheduling. In addition, it neglects other possible solutions. |
| Guo and Xuo [33] | List-based scheduling algorithm | CEFT | It efficiently compromises between cost and deadline in the course of scheduling. | It is a costly procedure, which is solely applicable to very faulty systems, because it sometimes reschedules to reach reliability. |
| Darbha and Agrawal [35] | Heuristic-based algorithm | Duplication | It increases parallelism by duplicating the execution of critical tasks on different VMs. | It may be costly, because it utilizes more resources and may charge users more money for residual VMs usage. |
| Palis et al. [36] | Heuristic-based algorithm | Clustering | It is a beneficial method for communication-intensive workflows to reduce data transmission costs. It may possibly reduce the total execution time. | If the degree of heterogeneity is high, the system performance degrades. |
| Hosseini Shirvani [8] | Meta-heuristic-based algorithm | GA-based | It efficiently explores the search space globally. | It does not utilize exploitation technique, which can potentially improve the final results. |
| Sujana et al. [19] | Meta-heuristic-based algorithm | PSO-based | It is a very fast approach. | It suffers from early convergence and usually gets stuck in local optima. |

posed, which suffered from early convergence [19]. The literature review is summarized in Table 1.

The review reveals that the majority of proposed models suffer from the lack of a balance between local and global searches during the course of optimization. In addition, improvement is still possible in exploring discrete search spaces. The current paper is aimed to fill the mentioned gaps.

## 3. Grey Wolf Optimizer (GWO)

The GWO mimics social hierarchy and predatory treatment of grey wolves. In social hierarchy of grey wolves, each wolf is categorized in one of the four groups. The first, second, and third bests are alpha ($\alpha$), beta ($\beta$), and delta ($\delta$), respectively; the remaining wolves are omega ($\omega$). In predating, the first three groups of wolves, namely $\alpha$, $\beta$, and $\delta$, are involved and the predation is done in three main stages, which are encircling the prey, hunting, and attacking the prey. The first two are done for exploration whereas the third stage is for exploitation in the search space. For encircling the prey, each individual $X(t)$ in the $t$th round of the optimization course finds its distance to a guessed prey $X_p(t)$. Then, it adjusts its path towards the prey. The distance value and the adjustment of direction are calculated via Eqs. (1) and (2):

$$\overrightarrow{D} = \left|\overrightarrow{C} \cdot \overrightarrow{X_P}(t) - \overrightarrow{X}(t)\right|, \tag{1}$$

$$\overrightarrow{X}(t+1) = \overrightarrow{X_P}(t) - \overrightarrow{A} \cdot \overrightarrow{D}. \tag{2}$$

To efficiently tune the optimization process in the search space, two vectors $\overrightarrow{A}$ and $\overrightarrow{C}$ are used, which are obtained by Eqs. (3) and (4). Recall, the first one is randomly and linearly changed whereas the second vector follows a completely random manner.

$$\overrightarrow{A} = 2\overrightarrow{a} \cdot \overrightarrow{r_1} - \overrightarrow{a}, \tag{3}$$

$$\overrightarrow{C} = 2\overrightarrow{r_2}. \tag{4}$$

As mentioned earlier, the elements of vector $\overrightarrow{a}$ are linearly declined from 2 to 0 during the course of optimization process and $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ are two real random vectors in the interval [0..1]. Then, each individual wolf adjusts its trajectory towards the hunt based on positions of wolves $\alpha$, $\beta$, and $\delta$ via Eq. (7).

$$\overrightarrow{X_a} = \left|\overrightarrow{C_1} \cdot \overrightarrow{X_a} - \overrightarrow{X}\right|, \qquad \overrightarrow{X_\beta} = \left|\overrightarrow{C_2} \cdot \overrightarrow{X_\beta} - \overrightarrow{X}\right|,$$

$$\overrightarrow{X_\delta} = \left|\overrightarrow{C_3} \cdot \overrightarrow{X_\delta} - \overrightarrow{X}\right|, \tag{5}$$

$$\overrightarrow{X_1} = \overrightarrow{X_a} - \overrightarrow{A_1} \cdot \left(\overrightarrow{D_a}\right), \qquad \overrightarrow{X_2} = \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot \left(\overrightarrow{D_\beta}\right),$$

$$\overrightarrow{X_3} = \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot \left(\overrightarrow{D_\delta}\right), \tag{6}$$

$$\overrightarrow{X}(t+1) = \frac{\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3}}{3}. \tag{7}$$

For attacking the prey (exploitation), the predatory process is finished by the attacking stage. This stage is performed when moving is ceased. The canonical GWO is customized to optimize the discrete workflow scheduling problem by leveraging new operators and *Walking Around* procedures.

## 4. Models and problem statement

Several models are presented before problem definition.

### 4.1. System and application models

The cornerstone of a cloud system is a datacenter. Each datacenter provides a list of heterogeneous VMs, $VM_{list} = VM_1, VM_2, \cdots, VM_q$. Each VM is determined in terms of a variable Million Instructions Per Second (MIPS). Figure 1 illustrates the proposed system model for cloud environment.

The users request workflow execution, which is received via the front end module of the cloud. Then, the scheduler, embedded in the cloud broker, assigns tasks to the available VMs to meet the user's demand. Workflow applications are modeled in DAGs. Each DAG contains nodes and arcs. A workflow $W$ contains $n$ tasks, $W = \{t_1, t_2, \cdots, t_n\}$, and set of arcs, $A = \{(t_i, t_j) | t_i, t_j \in W\}$. Every node is a task and an arc is used for data dependency between each pair of dependent tasks. A DAG has two specific nodes: *entry* and *exit*. The *entry* has no father while the *exit* has no child. Each task is assigned the number of Million Instructions (MIs). The processing time for execution of task $t_i$ on $VM_j$ is calculated by Eq. (8):

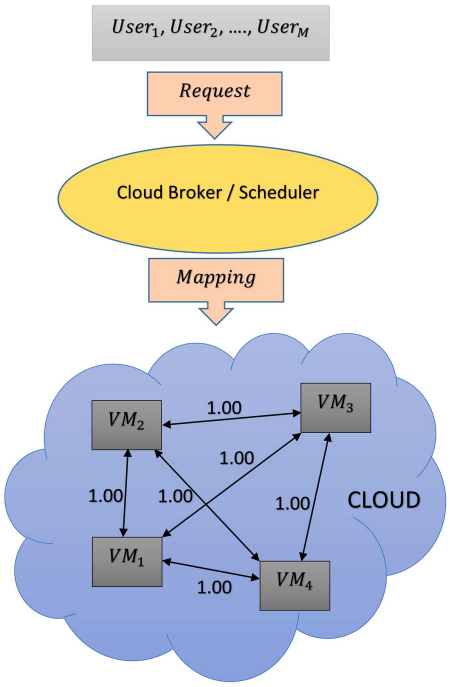


**Figure 1.** The proposed system model.

$$ET(t_i, VM_j) = \frac{\text{(MIS) assigned–to–a–task–}t_i}{\text{(MIPS) assigned–to–a–processor–}VM_j}. \tag{8}$$

The average amount of time needed for the execution of $t_i$ on the platform with $q$ available VMs is calculated by Eq. (9):

$$\overline{ET(t_i)} = \left(\frac{\sum_{j=1}^{q} ET(t_i, VM_j)}{q}\right). \tag{9}$$

The communication cost between each pair of dependent tasks in arc $e(t_i, t_j)$ is calculated by Eq. (10):

$$C(t_i, t_j) = \overline{L} + \frac{DV}{BW}. \tag{10}$$

The term $\bar{L}$ is relevant to the delay of the intrinsic link, which is negligible, and the term DV represents the data volume transferred in the network bandwidth (BW). If schedulers assign two dependent tasks to the same VM, the communication cost is omitted. Figure 2 depicts a DAG in which $t_1$ and $t_{10}$ are *entry* and *exit* nodes, respectively.

Table 2 presents the execution time for each task once on each of the three VMs of a heterogeneous platform. The last column shows the average execution time for each task measured by Eq. (9).

One important thing in scheduling is to use the Communication-to-Computation Rate ($CCR$) concept calculated by Eq. (11):

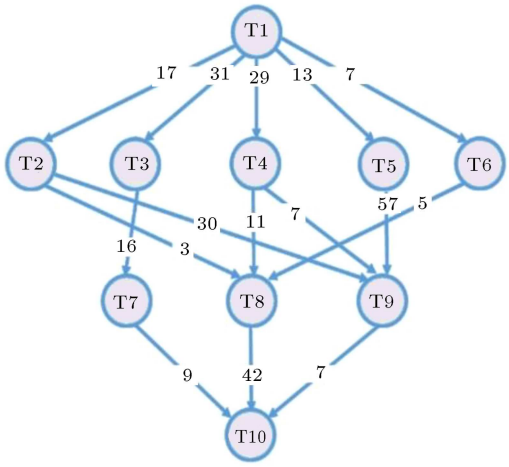


**Figure 2.** A typical workflow [11].

**Table 2.** Execution time on VMs [11].

| Task | VMs | | | $\overline{ET}$ |
|---|---|---|---|---|
| | $VM_1$ | $VM_2$ | $VM_3$ | |
| T1 | 22 | 21 | 36 | 26.3 |
| T2 | 22 | 18 | 18 | 19.3 |
| T3 | 32 | 27 | 43 | 34.0 |
| T4 | 7 | 10 | 4 | 7.0 |
| T5 | 29 | 27 | 35 | 30.3 |
| T6 | 26 | 17 | 24 | 22.3 |
| T7 | 14 | 25 | 30 | 23.0 |
| T8 | 29 | 23 | 36 | 29.3 |
| T9 | 15 | 21 | 8 | 14.7 |
| T10 | 13 | 16 | 33 | 20.7 |

$$CCR = \frac{\frac{1}{|A|}\left(\sum_{edge\,(t_i,t_j)}^{A} C(t_i,t_j)\right)}{\frac{1}{|W|}\left(\sum_{T_i}^{W} \overline{ET(t_i)}\right)}. \tag{11}$$

If the $CCR$ value is high, the given workflow is relatively communication-intensive. The value of CCR for a DAG in Figure 2 is 0.83 that means it is a moderate graph.

### *4.2. Scheduling model and problem formulation*

Task scheduling for workflow execution is a very important mission, because it determines to what available VM should be assigned which task. The proposed scheduling model follows two steps: prioritizing tasks and selecting VMs for the assignment of tasks [10]. In prioritizing the tasks, three ranking procedures are engaged that are upward, downward, and level ranking, each of which ranks tasks to produce its own valid tasks list. Eq. (12) through Eq. (16) are presented to provide different ordered tasks lists. Functions $Succ(t_i)$ and $Pred(t_i)$ determine immediate successor and predecessor tasks of a given task $t_i$, respectively. Upward ranking recursively starts from *exit* node to reach *entry* node. It calculates ranking value for the *exit* node by Eq. (12), but for other nodes, the ranking values are calculated by Eq. (13):

$$UpRank(t_{exit}) = \overline{ET(t_{exit})}, \tag{12}$$

$$UpRank(t_i) = \overline{ET(t_i)} + \max_{\forall\ t_j \in succ\,(t_i)} \Big\{ UpRank(t_j) + C(t_i,t_j) \Big\}. \tag{13}$$

Downward ranking starts from *entry* node to reach *exit* node. It calculates ranking value for the *entry* node by Eq. (14), but for other nodes, the ranking values are calculated by Eq. (15):

$$DownRank(t_{entry}) = 0, \tag{14}$$

$$DownRank(t_i) = \max_{\forall\ t_j \in \mathrm{pred}\,(t_i)} \Big\{ DownRank(t_j) + \overline{ET(t_j)} + \overline{C(t_j,t_i)} \Big\}. \tag{15}$$

Finally, the level ranking procedure starts from *entry* to reach *exit*. It assigns the level ranking value of zero to the entry, but for others, the values are calculated by Eq. (16):

$$LevelRank(t_i) = \max_{\forall\ t_j \in pred\,(t_i)} \{LevelRank(t_j)\} + 1. \tag{16}$$

For each ranking, the tasks are sorted based on rank labels assigned to each task. The sorting is ascending order for both downward and level rankings whereas it is descending for upward ranking.

To select VM, the $EFT$ and Earliest Start Time ($EST$) functions are exploited. The $EFT\,(t_i, VM_j)$ function determines the earliest time that the virtual machine $VM_j$ finishes execution of $t_i$ provided it is assigned to this VM. The function $EST\,(t_i, VM_j)$ determines the earliest possible time for beginning the execution of $t_i$ on virtual machine $VM_j$. This function takes both the availability time and the prerequisites for receiving the data of a given task $t_i$ for $VM_j$ into account. For entry and non-entry tasks, $EST(t_i, VM_j)$ is calculated by Eqs. (17) and (18), respectively.

$$EST(t_{entry}, VM_j) = 0, \tag{17}$$

$$EST(t_i, VM_j) = \max\Big\{ Avail(VM_j), \max_{\forall\ t_q \in pred\,(t_i)} \Big\{ AFT(t_q) \Big\} + \overline{C(t_q,t_i)} \Big\}. \tag{18}$$

In Eq. (18), the maximum value between $Avail(VM_j)$ and the latest time that the prerequisite data of $t_i$ is received must be considered. The term $Avail(VM_j)$ indicates the earliest time that VM is free to do a new mission. The term $AFT(t_q)$ is elaborated in Eq. (19) to indicate the actual finish time of task $t_i$ on the available VM guaranteeing the EFT. The term *index* indicates the number of VMs in the *VMList* that return the minimum value:

$$AFT(t_q, VM_{index}) = \min_{\forall\ VM_j \in VMList} \{EFT(t_q, VM_j)\}. \quad (19)$$

In addition, the $EFT(t_i, VM_j)$ is calculated by the summation of the two values of $EST(t_i, VM_j)$ and $ET(t_i, VM_j)$ drawn in Eq. (20):

$$EFT(t_i, VM_j) = EST(t_i, VM_j) + ET(t_i, VM_j). \quad (20)$$

The total execution time (*makespan*) is calculated by Eq. (21). This is the objective function that workflow scheduling algorithm tries to minimize:

$$makespan = \min \left\{ \max_{\forall t_i \in W} (AFT(t_i)) \right\}. \quad (21)$$

Since the existing schedulers present a limited number of valid tasks lists, there is a clear need for finding optimal solutions by efficiently exploring the search space. Therefore, D-GWO is developed to bridge the gap.

## 5. An illustrative example

An illustrative example shows the effectiveness of D-GWO in workflow scheduling. Figure 2 is considered as a case study. D-GWO is compared with other state-of-the-arts. The comparatives are two famous list schedulers of HEFT [10] and PEFT [11], two meta-heuristics of Multiple-Priority Queues and Genetic Algorithm (MPQGA) [5] and a Customized Simulated Annealing (C-SA) [25], and a hybrid D-PSO [38]. Table 3 shows the rank values of each task derived by each algorithm. Table 4 shows lists of tasks generated by different approaches along with the gained final *makespan*. Figure 3 illustrates the performance of D-GWO against others works in the literature. It proves that D-GWO dominates others in terms of *makespan*.

## 6. Proposed discrete grey wolf optimization algorithm for solving the workflow scheduling problem

A novel D-GWO is presented to solve the discrete workflow scheduling problem. To this end, the elementary concepts, new operators, and procedures are introduced.

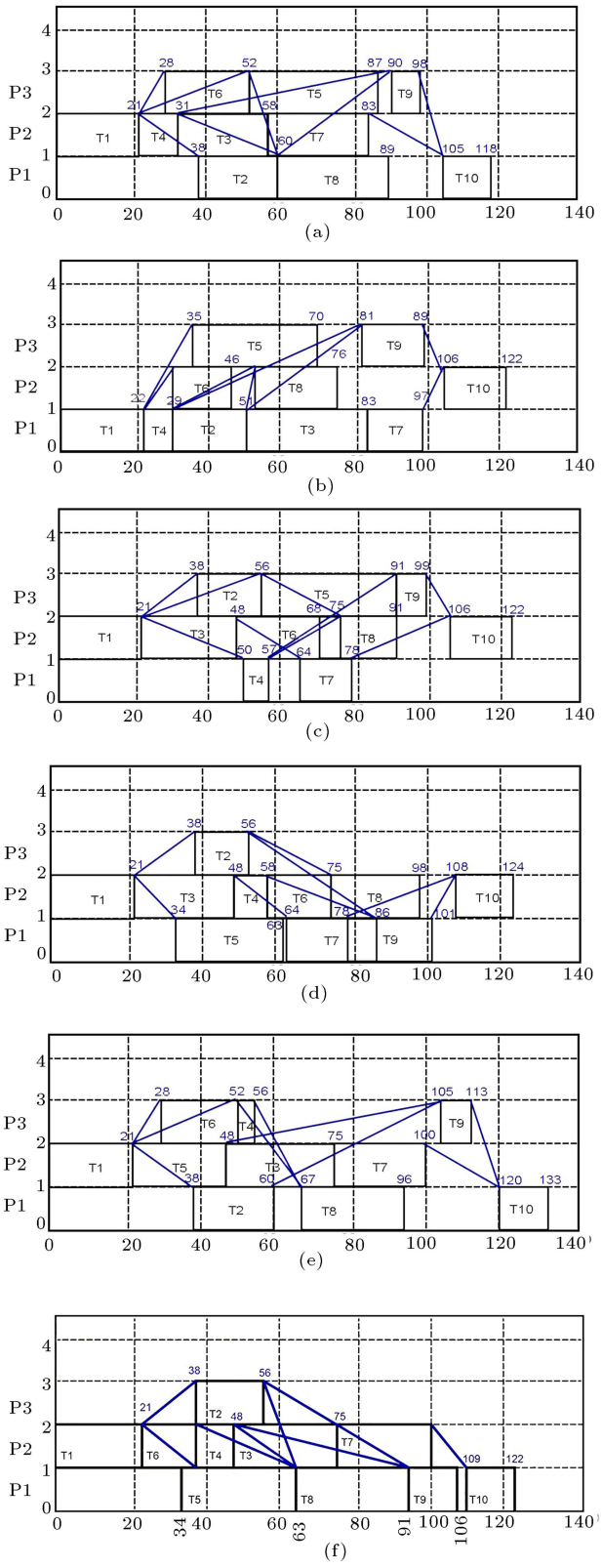


**Figure 3.** An illustrative example: (a) D-GWO scheduler, $makespan = 118$, (b) PEFT scheduler [11], $makespan = 122$, (c) MPQGA scheduler [5], $makespan = 122$, (d) S-CA scheduler [23], $makespan = 124$, (e) HEFT-upward scheduler [10], $makespan = 133$, and (f) D-PSO scheduler [42], $makespan = 122$.

**Table 3.** Ranking value assigned to each task in different list schedulers.

| Tasks | Ranking | | | |
|---|---|---|---|---|
| | **PEFT** | ***Upward*** | ***Downward*** | **Level** |
| T1 | 72.7 | 169.0 | 0.0 | 0 |
| T2 | 41.0 | 114.3 | 43.3 | 1 |
| T3 | 37.0 | 102.7 | 57.3 | 1 |
| T4 | 43.7 | 110.0 | 55.3 | 1 |
| T5 | 31.0 | 129.7 | 39.3 | 1 |
| T6 | 41.7 | 119.3 | 33.3 | 1 |
| T7 | 17.0 | 52.7 | 107.3 | 2 |
| T8 | 20.7 | 92.0 | 73.3 | 2 |
| T9 | 16.3 | 42.3 | 126.6 | 2 |
| T10 | 0 | 20.7 | 148.3 | 3 |

### *6.1. Problem encoding (memetic and wolf representation)*

The problem encoding phase is one of the most important issues that has impact on tracking and performance of a proposed meta-heuristic algorithm [39–41]. The task name is considered as an allele. Hence, genes are selected from a set of integer numbers $\{1, 2, \cdots, n\}$ that are task numbers. A wolf (as a candidate solution) is an ordered $n$ number of non-identical tasks. For instance, a valid list of tasks $List_{D\text{-}GWO} = \{t_1, t_4, t_3, t_6, t_7, t_2, t_5, t_9, t_8, t_{10}\}$ is encoded to a wolf, as illustrated in Figure 4.

### *6.2. Auxiliary binary vectors and operators*

The trajectory of an individual wolf toward a prey is conducted by three best wolves, namely the alpha ($W_\alpha$), beta ($W_\beta$), and delta ($W_\delta$). Therefore, new auxiliary binary vectors and operators are proposed

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| A wolf representation | 1 | 4 | 3 | 6 | 7 | 2 | 5 | 9 | 8 | 10 |

**Figure 4.** An encoded wolf.

to take benefit of the knowledge of the leader wolves about the traversed discrete search space. Thus, binary vectors $Token_i = (b_1, b_2, \cdots, b_n)$ and $Adjuster_i = (a_1, a_2, \cdots, a_n)$ are applied for comparison between each agent and leader wolves. In this regard, each wolf must relocate tasks in the task list similar to the encodings of the leaders. The zero bit means that the corresponding task is not required to change. Note that $t_{entry}$ and $t_{exit}$ are not to be changed. Hence, they are set to zero in the *Token* vector. Furthermore, in the initialization, all tasks are to be changed, which is why the value of the *Token* vector is the initialized one.

#### *6.2.1. Binary operators \ and ⊗*

The operator \ is used to indicate the differences in the corresponding tasks of two wolves. For instance, take $n = 6$, $Token_1 = (0, 1, 1, 0, 1, 0)$ for $W_1$, and $Token_\alpha = (0, 0, 1, 1, 1, 0)$ for $W_\alpha$. Then, $Token_1 \backslash Token_\alpha = (0, 1, 0, 1, 0, 0)$, that is, the output of the same bit is zero, because it does not require to be changed. For the operator ⊗, the associated bit is changed provided the corresponding *Adjuster* value is one. If $Token_1 = (0, 1, 1, 0, 0, 0)$ and $Adjuster_1 = (0, 0, 1, 0, 1, 0)$, then $Token_1 = Token_1 \otimes Adjuster_1 = (0, 1, 1, 0, 0, 0) \otimes (0, 0, 1, 0, 1, 0) = (0, 1, 0, 0, 1, 0)$. The adjuster vector is a clue for the *Token* vector to close off duplicate changes on especial tasks.

Recall, if the value of $b_j$ in *Token* is one, the associated task in the list of a wolf $W_i$ can be arbitrarily substituted with a task with the corresponding binary $b_k$ value of one. This substitution is done by a heuristic. The value of zero means no change is required. The corresponding *Adjuster* value is used to change the

**Table 4.** Ordered list of tasks produced by comparative algorithms along with the final *makespan*.

| No. | Approach/Ref. | Generated list of tasks | Final *makespan* |
|---|---|---|---|
| 1 | HEFT-upward [10] | $\{t_1, t_5, t_6, t_2, t_4, t_3, t_8, t_7, t_9, t_{10}\}$ | 133 |
| 2 | HEFT-downward [10] | $\{t_1, t_6, t_5, t_2, t_4, t_3, t_8, t_7, t_9, t_{10}\}$ | 136 |
| 3 | HEFT-level [10] | $\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ | 143 |
| 4 | PEFT [11] | $\{t_1, t_4, t_6, t_2, t_3, t_5, t_8, t_7, t_9, t_{10}\}$ | 122 |
| 5 | MPQGA [5] | $\{t_1, t_6, t_5, t_4, t_2, t_3, t_7, t_8, t_9, t_{10}\}$ | 122 |
| 6 | Customized-SA (CSA) [23] | $\{t_1, t_3, t_5, t_2, t_4, t_6, t_7, t_8, t_9, t_{10}\}$ | 124 |
| 7 | D-PSO [42] | $\{t_1, t_6, t_5, t_4, t_2, t_3, t_8, t_7, t_9, t_{10}\}$ | 122 |
| 8 | Proposed D-GWO | $\{t_1, t_4, t_3, t_6, t_7, t_2, t_5, t_9, t_8, t_{10}\}$ | 118 |

value of *Token* for the next round. This change is a clue to not modify the corresponding task again in the next round.

### 6.3. D-GWO algorithm description

Algorithm 1 starts with the initial population by calling Algorithm 2 that utilizes the theorems in [1]. The theorems allow to permute tasks of the same level in a list. The vectors *Token* and *Adjuster* are set to $\overrightarrow{1}$, which means all tasks are to be changed. Then, each wolf is evaluated by a fitness calculated by Algorithm 3. The first, second, and third bests are known as the $\alpha$, $\beta$, and $\delta$ wolves. The best so far solution is kept in as a possible optimal solution. The main loop of Algorithm 1 starts between lines 8 through 31. It is iterated until the termination criterion is met. In each iteration, for each wolf, some operations are performed. Firstly, Algorithm 4 is called to encircle the prey for exploration. If the change made upon a given wolf is valid, the best so far solution can be updated; otherwise, the change of the wolf is retreated. Also, the *Token* is updated based on the *Adjuster* and the *Adjuster* is updated in Algorithm 4 to preclude duplicate changes. Afterwards, the exploitation phase is

**Input**:
A given DAG application with its specification
VMs: $\{VM_1, VM_2, \ldots, VM_M\}$
n : number of tasks in a given DAG
M : number of VMs
$m$ : number of wolves
*MaxIteration* : Maximum of iterations
**Output**:
An optimal task scheduling solution

**1**: Call **Algorithm 2** to generate an initial population (* each wolf $W_i$=($w_{i1}$, $w_{i2}$,…, $w_{in}$) where $i$=1,2,…,$m$ and each field $w_{ij} = t_k$ is a task in a given DAG. *)
**2**: Initialize two binary vectors $Token_i$=($b_{i1}$, $b_{i2}$,…, $b_{in}$)=$\overrightarrow{1}$ and $Adjuster_i$=($a_{i1}$, $a_{i2}$,…, $a_{in}$)=$\overrightarrow{1}$. (* all tasks initially are to be exchanged *)
**3**: Call **Algorithm 3** to calculate the fitness for each $W_i$;
**4**: Let $W_\alpha$ be the first best wolf, $Token_\alpha$ and $Adjuster_\alpha$ be associated Token and Adjuster of alpha wolf.
**5**: Let $W_\beta$ be the second best wolf, $Token_\beta$ and $Adjuster_\beta$ be associated Token and Adjuster of beta wolf.
**6**: Let $W_\delta$ be the third best wolf, $Token_\delta$ and $Adjuster_\delta$ be associated Token and Adjuster of delta wolf.
**7**: Let BestSoFar ← $W_\alpha$ ; BestSolution ← Fitness($W_\alpha$)
**8**: **while** the termination criteria is not met **do**
**9**: **for** each wolf $W_i$ in population where i=1,…, $m$ **do**
----------(* Exploration *)----------
**10**: Call **Algorithm 4** for encircling the prey ( update the position $W_i$ towards the prey)
**11**: **if Validate** ($W_i$) is True **then**
**12**: **if** Fitness($W_i$) is better than BestSolution **then**
**13**: BestSoFar ← $W_i$
**14**: BestSolution ← Fitness($W_i$)
**15**: **end-if**
(* Update $Token_i$ based on $Adjuster_i$ for next round usage *)
**16**: $Token_i = Token_i \otimes Adjuster_i$ ;
----------(* Exploitation *)----------
**17**: Draw integer $q$ ~ [1..3] for Walking Around procedures
**18**: **if** R=1 **then**
**19**: Call **Algorithm 5** for $Permutation_1(W_i)$
**20**: **elseif** if R=2 **then**
**21**: Call **Algorithm 6** for $Permutation_2(W_i)$
**22**: **else**
**23**: Call **Algorithm 7** for $Permutation_3(W_i)$
**24**: **end-if**
**25**: **else**
**26**: Retreat $W_i$
**27**: **end-if**
**28**: Call **Algorithm 3** to calculate the fitness for each $W_i$;
**29**: Let new $W_\alpha$, $W_\beta$, and $W_\delta$ three best wolves based on updated fitness values
**30**: **end for**
**31**: **end while**
**32**: return *BestSoFar* and *BestSolution*
**33**: **End** {**Algorithm 1**}

**Algorithm 1.** D-GWO-scheduler.

**Input:**
G : A given DAG with its specifications
$n$ : number of tasks
$m$ : number of wolves in population
**Output:**
W : A semi-conducted random wolves
(* Note that, W=($W_1, W_2, \ldots, W_m$) where each $i$-th wolf is $W_i$=($W_{i1}, W_{i2}, \ldots, W_{in}$) *)

**1**: $W_1$ ← Call upward ranking (G) based on **Eqs. (12-13).**
**2**: $W_2$ ← Call downward ranking (G) based on **Eqs. (14-15).**
**3**: $W_3$ ← Call level ranking (G) based on **Eq. (16).**
**4**: $W_4$ ← Call PEFT (G) based on **Algorithm** in Ref. [11].
**5**: Calculate depth of given DAG in D=Level (G) based on **Eq. (16).**
**6**: Calculate List=$\{t_1, SubList_1, SubList_2, \ldots, SubList_{D-1}, t_n\}$ where $SubList_j$ having nodes in the same Level=j.
**7**: **For** i=5 to $m$ **do**
**8**: $W_i$ ← A List getting from permutation of tasks in $SubList_j$ [1].
**9**: **End-For**
**10**: return W
**11**: **End** {**Algorithm 2**}

**Algorithm 2.** Initial wolves.

**Input:**
$W_i$ : A wolf;
$n$ : number of tasks in W;
VMs : $\{VM_1, VM_2, \ldots, VM_M\}$
**Output:**
An assignment and *makespan* value

**1**: **While** there exists an unscheduled task in ordered list **Do**
**2**: Select an unscheduled task $t_j$ from chromosome (list)
**3**: **For** each $VM_k$ in VMs list **Do**
**4**: Calculate EFT($t_j, VM_k$) based on **Eq. (20)**.
**5**: Assign task $t_j$ to $VM_k$ that returns minimum EFT ($t_j, VM_k$).
**6**: **End-For**
**7**: return *makespan*=AFT($t_{Exit}$)
**8**: **End** {**Algorithm 3**}

**Algorithm 3.** Fitness function.

**Input:**
$W_i, W_\alpha, W_\beta, W_\delta$ : Wolf;
$n$ : number of tasks in a wolf
$Token_i, Token_\alpha, Token_\beta, Token_\delta$ : Token;
$Adjuster_i, Adjuster_\alpha, Adjuster_\beta, Adjuster_\delta$ : $Adjuster$;
VMs : $\{VM_1, VM_2, \ldots, VM_M\}$
**Output:**
A new updated wolf $W_i$

**1**: Let $F_1$=Fitness ($W_\alpha$), $F_2$=Fitness ($W_\beta$), and $F_3$=Fitness ($W_\delta$)
(* $F_1 \le F_2 \le F_3$ *)
**2**: Let $P_1=\frac{F_1}{F_1+F_2+F_3}$, $P_2=\frac{F_2}{F_1+F_2+F_3}$, and $P_3=\frac{F_3}{F_1+F_2+F_3}$
(* $P_1 \le P_2 \le P_3$ and $P_1 + P_2 + P_3 = 1$ *)
**3**: $TokenDiff_1 = Token_i \setminus Token_\alpha$
**4**: $TokenDiff_2 = Token_i \setminus Token_\beta$
**5**: $TokenDiff_3 = Token_i \setminus Token_\delta$
**6**: **For** j=1 To $n$ **Do**
**7**: draw $q \sim (0,1)$
**8**: **if** $q < P_1$ **Then**
**9**: $Adjuster_i$(j) ← $TokenDiff_1$(j)
**10**: **elseif** $q < P_2$ **Then**
**11**: $Adjuster_i$(j) ← $TokenDiff_2$(j)
**12**: **else**
**13**: $Adjuster_i$(j) ← $TokenDiff_3$(j)
**14**: **end-if**
**15**: **End-For**
**16**: $Adjuster_i(1) = Adjuster_i(n) = 0$;
**17**: Update $W_i$ for encircling the prey by updating its trajectory following $Adjuster_i$
**18**: (* Exchange $t_j$ and $t_k$ where $Adjuster_i(t_j) = Adjuster_i(t_k) = 1$ and $t_j \neq t_k$ *)
**19**: return $W_i$
**20**: **End** {**Algorithm 4**}

**Algorithm 4.** Encircling the prey.

started. The *Walking Around* can potentially improve the solutions from the exploration phase. To this end, a random integer is drawn to call one of the *Walking Around* procedures casually. In the *Walking Around* process, procedures which make permutation are introduced. These procedures are well defined in such a way to permute the search space efficiently. After the exploitation phase is done and changes happen, all of the wolves are evaluated again to find three new leaders of $\alpha$, $\beta$, and $\delta$ wolves. Finally, the best so far solution is returned. Algorithm 2 generates a semi-random initial population. It utilizes theorems of Ref. [1], which produce promising individuals. Algorithm 3 calculates the fitness function. Algorithm 4 adopts the position information of individual and leader wolves: $\alpha$, $\beta$, and $\delta$. Moreover, their *Token* and *Adjuster* vectors are considered as input. Based on the fitness of the leaders, three probability parameters $P_1$, $P_2$, and $P_3$ are calculated to get a chance for adopting parts of the knowledge of the leader wolves about the search space for an individual wolf to encircle the prey. Firstly, the difference *Token* (*TokenDiff*) between the *Token* of $W_i$ and those of other leaders $W_\alpha$, $W_\beta$, and $W_\delta$ is obtained. For each task in the list, it is randomly determined which part is drawn from which leader wolf. Afterwards, the update trajectory is performed by incorporating the *Adjuster* vector. The value zero used in line #16 means the *entry* and *exit* tasks are not to be changed. In line #17, the tasks with corresponding bit values equal to one in the *Adjuster* are candidates for exchange. After exchange, a new individual is generated as the output.

*6.3.1. Walking around procedures (for Exploitation)*

Here, *Walking Around* procedures for a given solution are introduced. Three procedures, which permute the discrete search space, are $Permutation_1$, $Permutation_2$, and $Permutation_3$. Among the mentioned three *Walking Around* procedures, one is randomly called. Algorithm 5 presents $Permutation_1$ procedure along with its application. Algorithm 5 as the first kind of permutation randomly opts a meme $W[i]$. Then, it quests for finding its first successor task in the list such as $W[j]$. A selected meme $W[i]$ must be exchanged by $W[k]$ in which $k \in [i+1..j-1]$ so that the last predecessor of meme $W[k]$ is ahead of $W[i]$ in the ordered list. It definitely keeps the topological sorting attribute. Figure 5 illustrates how $Permutation_1$ performs. It randomly draws $W[4] = t_2$. The first successor of $t_2$ is $W[7] = t_9$. Algorithm 5 finds $k \in [5..6]$ so that the last predecessor task of $W[k]$ is ahead of $W[4]$. In this case, the last predecessor task of $W[6]$, which is $t_1$, is ahead of $W[4]$. The reason for choosing $W[6]$ is to exchange it with $W[4]$. The new wolf is depicted in Figure 5. Algorithm 6 presents $Permutation_2$ procedure. Algorithm 6 as the second kind of permutation randomly opts a meme $W[i]$ in a wolf $W$. Then, it quests for finding its last predecessor task in the list such as $W[j]$. A selected meme $W[i]$ must be exchanged by $W[k]$ in which $k \in [j+1..i-1]$ so that the first successor of meme $W[k]$ is placed after of $W[i]$ in the ordered list. It definitely keeps the topological sorting attribute. Figure 6 illustrates

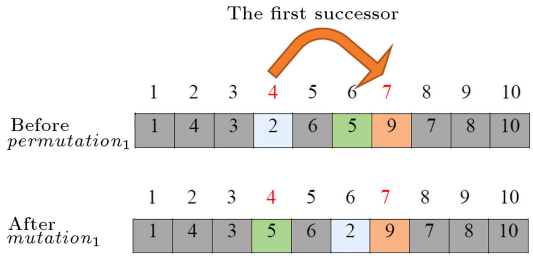


**Figure 5.** Performance of $Permutation_1$.

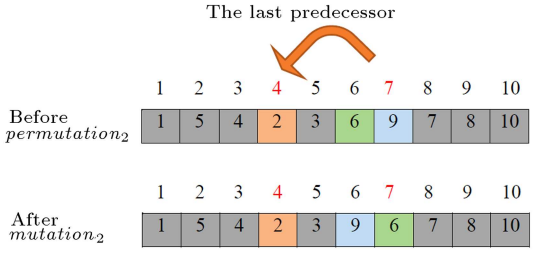


**Figure 6.** Performance of $Permutation_2$.

**Input:**
$W_i$ [1.. $n$] : A wolf
$n$ : number of tasks in a wolf
**Output:**
$W_i$ [1.. $n$] : A modified wolf

**1**: Draw a random integer R in ~ [2..$n$-1]
**2**: Z ← $W_i$ [R]
**3**: $t_y$ ← Find task $t_y$ in set { $Succ(t_Z)$ } which appears ahead in the $W_i$ [1..$n$] where $y$=$W_i$ [S]
**4**: Find random $j$ ∈ [R+1..S-1] and $t_q$=$W_i$ [$j$]; so that the last item of $Pred(t_q)$ appeared in the wolf is ahead of task $t_z$ where $z$=$W_i$ [R]
**5**: Exchange ($W_i$ [R], $W_i$ [$j$])
**6**: return $W_i$ [1.. $n$]
**7**: **End** {**Algorithm 5**}

**Algorithm 5.** $Permutation_1$ procedure.

**Input:**
$W_i$ [1.. $n$] : A wolf
$n$ : number of tasks in a wolf
**Output:**
$W_i$ [1.. $n$] : A modified wolf

**1**: Draw a random integer R in ~ [2..$n$-1]
**2**: Z ← $W_i$[R]
**3**: $t_y$ ← Find task $t_y$ in set { $Pred(t_Z)$ } which appears latter in the $W_i$[1..$n$] where $y$=$W_i$[S]
**4**: Find random index $j$ ϵ [S+1..R-1] and $t_p$=$W_i$ [$j$]; so that the first item of $Succ(t_p)$ appeared in the wolf is latter of task $t_z$ where $z$=$W_i$ [R]
**5**: Exchange ($W_i$ [R], $W_i$ [j])
**6**: return $W_i$ [1.. $n$]
**7**: **End** {**Algorithm 6**}

**Algorithm 6.** *Permutation*$_2$ procedure.

**Input:** G : is a given DAG along with its specifications
$W_i$ [1.. $n$] : A wolf
$n$ : number of tasks in a wolf
**Output:**
$W_i$ [1.. $n$] : A modified wolf

**1**: L=find graph G's level by Level (G) in **Eq.** (16).
**2**: Find two independent nodes $t_x$= $W_i$ [$j$] and $t_y$= $W_i$ [$k$] in consecutive level; so, Level($t_x$)≠Level($t_y$)
**3**: Exchange($W_i$ [$j$], $W_i$ [$k$]) /* $t_x$←→$t_y$
**4**: **if** $W_i$ [1.. $n$] is not valid **then**
**5**: return the input intact $W_i$ [1.. $n$];
**6**: **End-if**
**7**: return $W_i$ [1.. $n$]
**8**: **End** {**Algorithm 7**}

**Algorithm 7.** *Permutation*$_3$ procedure.

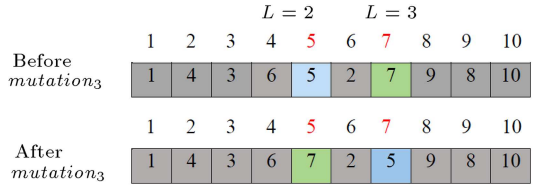


**Figure 7.** Performance of *Permutation*$_3$.

how *Permutation*$_2$ acts. It randomly draws $W[7] = t_9$. The last predecessor of $t_9$ is $W[4] = t_2$. Algorithm 6 finds $k \in [5..6]$, so that the first successor task of $W[k]$ is placed after $W[7]$. In this case, the first successor task of $W[6]$, which is $t_8$, is placed after of $W[7]$. The reason for selecting $W[6]$ is to exchange it with $W[7]$. The new wolf is depicted in Figure 6. Algorithm 7 presents *Permutation*$_3$ procedure. Algorithm 7 as the third kind of permutation firstly measures the maximum level of DAG. Then, it randomly opts for two independent memes that are associated with two different levels. The independent tasks are substituted. If the newborn wolf is valid, it is definitely returned. Otherwise, the first wolf without change is returned. Figure 7 illustrates how *Permutation*$_3$ works. Two memes $W[5] = t_5$ and $W[7] = t_7$ are independent tasks belonging to two different levels $L = 2$ and $L = 3$, respectively. Calling *Permutation*$_3$ improves makespan from 138 to 118, as depicted in Figure 7.

## 7. Performance evaluation

To evaluate the performance of the D-GWO, evaluation parameters, dataset, and settings are presented.

### 7.1. Evaluation parameters

The famous scheduling evaluation metrics are *makespan*, SLR, *speedup*, and *efficiency*. The important QoS parameter that the user endures is *makespan*, calculated by Eq. (21). However, utilizing only *makespan* does not indicate how efficiently the scheduling works. Therefore, the *makepan* must be compared with the Critical Path (CP) of a given DAG. The CP is the longest serial path, which is not parallelizable. Therefore, the *makespan* is usually longer than this length. This is why the new parameter of Scheduling Length Ratio (SLR) is introduced, which is calculated by Eq. (22):

$$SLR = \frac{makespan}{\sum_{t_j \in CriticalPath} \min\left(W((t_i, VM_k)) \forall\, VM_k \in VMList\right)}. \quad (22)$$

Another important parameter is to compute how the proposed model makes *speedup*. The *speedup* value measured by Eq. (23) means the reverse relative parallel execution time against serial execution time:

**Table 5.** Different simulation datasets.

| CCR | Communication cost | Computation cost | Graph type |
|---|---|---|---|
| 0.5 | [2..10] | [2..15] | Computation-intensive |
| 1.0 | [2..10] | [2..10] | Moderate |
| 5.0 | [5..20] | [2..5] | Rather communication-intensive |
| 10.0 | [10..40] | [2..5] | Communication-intensive |

$$speedup = \frac{\text{Serial-execution-on-}VM}{makespan} = \frac{\min\limits_{\forall\ VM_k \in VMList} \left\{ \sum\limits_{\forall t_j \in \{T\}} W(t_i, VM_k) \right\}}{makespan}. \quad (23)$$

The *speedup* does not show how many processors are involved in gaining *speedup*. This is why the auxiliary parameter *efficiency* is introduced, which is calculated by Eq. (24):

$$efficiency = \frac{speedup}{\text{Number-of-Used-}VMs} * 100\%. \quad (24)$$

### 7.2. Dataset

The molecular dynamics depicted in Figure 8 is one of the most important scientific workflows that is pervasively used in physics branches [10,17,42]. It is a well-suited workflow for testing because of its unstructured and unbalanced shape. To efficient evaluation, several datasets are produced to generate different workflows with different attributes. Therefore, different molecular workflows are generated for the simulation datasets. Table 5 elaborates the datasets.

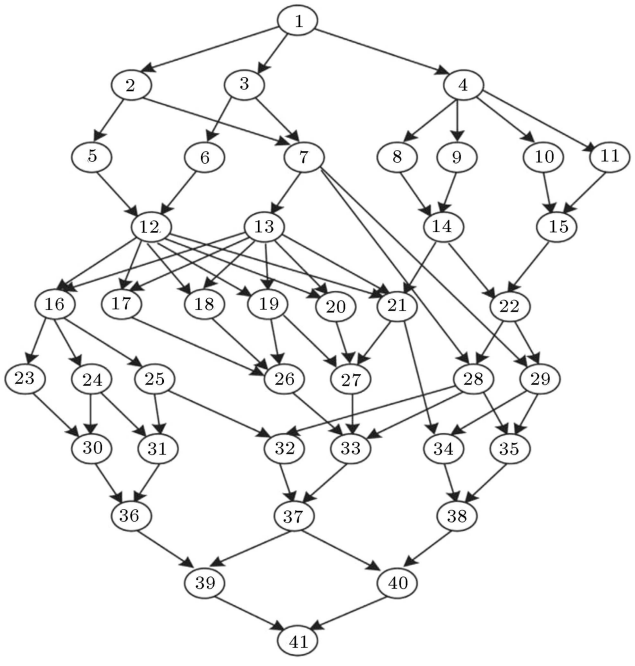


**Figure 8.** Molecular workflow [10,17,36].

The width of the given workflow is 7, because the maximum available tasks in each level are 7. Thus, utilizing more than 7 VMs does not have any affection on the final results [1,8]. For each scenario, the number of utilized VMs is 3, 5, and 7.

### 7.3. Results and discussion

This section clarifies simulation settings and discusses the results.

#### 7.3.1. Parameter settings and performance analysis

The proposed D-GWO scheduler was compared with several schedulers existing in the literature. To this end, the most efficient ones were selected, namely PEFT [11], MPQGA [5], C-SA [25], and D-PSO [38]. All experiments were executed in fair conditions on the same platform. Table 6 depicts the parameter setting of each.

Each scenario was independently executed 20 times. The average results were reported in terms of *makespan*, *SLR*, *speedup*, and *efficiency*. Figure 9 illustrates that D-GWO beats other state-of-the-arts in term of *makespan* in all scenarios.

Table 7 elaborates the comparisons. The Relative Percentage Deviation (RPD) concept is applied in order to stipulate the amount of improvement with the proposed model [1]. As Table 7 shows, following D-GWO as the best, D-PSO competes MPQGA in some cases, but in the majority of cases MPQGA works better. In summary, they work the same in 7 scenarios; in one scenario, D-PSO works better; and in the remaining 4 scenarios, the MPQGA performs better than D-PSO. Totally, after D-GWO, the MPQGA, D-PSO, C-SA, and PEFT are known from better to the worst.

Figure 10 depicts the comparison of D-GWO with other state-of-the-arts in terms of SLR, which is a normalized parameter regardless of graph shape and size. Again, in all scenarios, D-GWO outperforms others in terms of SLR. After D-GWO as the best, the MPQGA, D-PSO, C-SA, and PEFT are placed from the second to the fifth best.

Table 8 is dedicated to elaborate the information

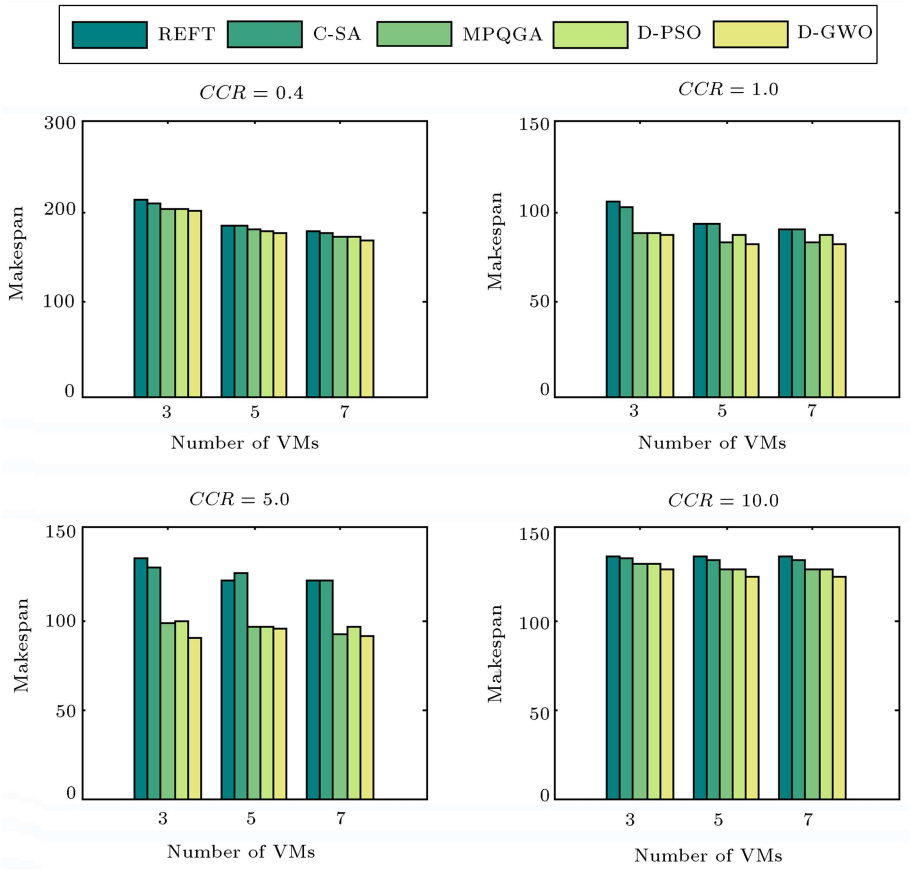


**Figure 9.** Performance comparison of D-GWO with others in terms of *makespan*.

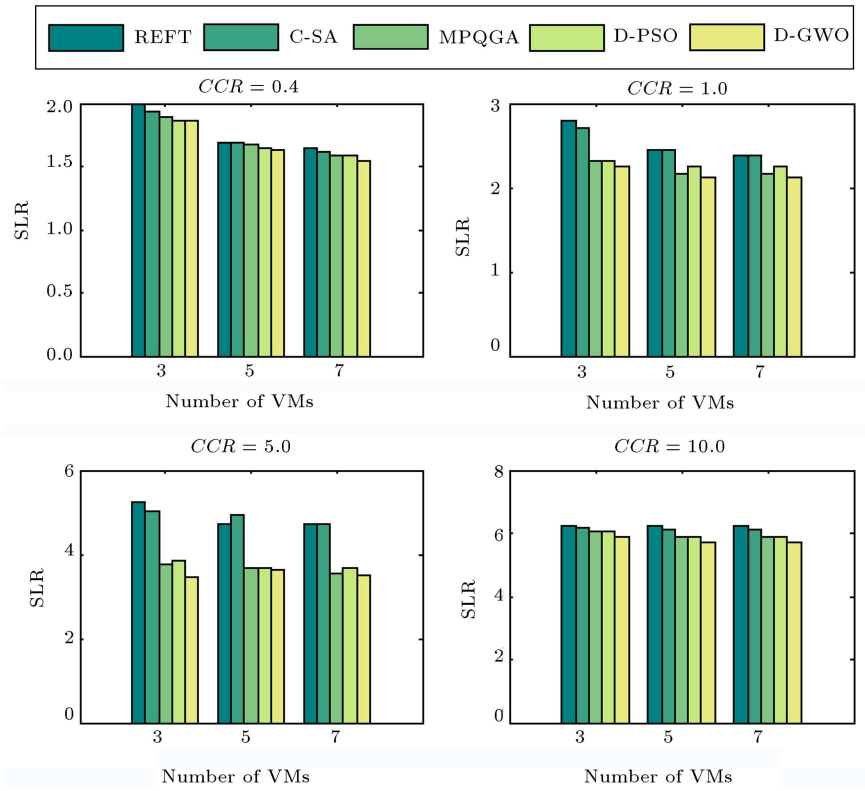


**Figure 10.** Performance comparison of D-GWO with others in terms of *SLR*.

**Table 6.** Parameter settings of the comparative algorithms.

| Algorithms | Specific parameters | | Population size | Max iterations |
|---|---|---|---|---|
| | Parameter | Value | | |
| MPQGA [5] | Crossover percentage:<br>Mutation percentage: | 0.8<br>0.05 | 50 ∼ 150 depends on scenario | 100 ∼ 150 depends on scenario |
| PEFT [11] | Fixed heuristic | Original settings | NA | One time |
| C-SA [23] | Freeze<br>$T_0$<br>$\Delta T$ | 0<br>1000<br>20 | Point-wise | 10 ∼ 20 iterations in each temperature depends on scenario |
| D-PSO [42] | $C_1 = C_2$<br>$\omega$<br>$V_{\max}$ | 1.5<br>1.2<br>4 | 50 ∼ 100 depends on scenario | 50 ∼ 150 depends on scenario |

$T_0$: Initial temperature; $\Delta T$: Decrease in the amount of temperature; Freeze: Freeze temperature for the final condition; $V_{\max}$: Velocity limit for clamping; $C_1$: Personal acceleration coefficient; $C_2$: Social acceleration coefficient; $\omega$: Inertia coefficient

**Table 7.** Comparison of the literature in terms of *makespan*.

| CCR | No. of VMs | *Makespan* | | | | | RPD (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | PEFT | C-SA | MPQGA | D-PSO | D-GWO | PEFT | C-SA | MPQGA | D-PSO |
| 0.4 | 3 | 215 | 210 | 204 | 204 | **201** | 6.51 | 4.29 | 1.47 | 1.47 |
| | 5 | 183 | 183 | 180 | 178 | **176** | 3.83 | 3.83 | 2.22 | 1.12 |
| | 7 | 178 | 175 | 171 | 171 | **167** | 6.18 | 4.57 | 2.34 | 2.34 |
| 1.0 | 3 | 106 | 103 | 88 | 88 | **86** | 6.18 | 4.57 | 2.34 | 2.34 |
| | 5 | 93 | 93 | 82 | 86 | **81** | 12.90 | 12.90 | 1.22 | 5.81 |
| | 7 | 90 | 90 | 82 | 86 | **81** | 10.00 | 10.00 | 1.22 | 5.81 |
| 5.0 | 3 | 136 | 131 | 98 | 100 | **90** | 33.82 | 31.30 | 8.16 | 10.00 |
| | 5 | 123 | 128 | 96 | 96 | **95** | 22.76 | 25.78 | 1.04 | 1.04 |
| | 7 | 123 | 123 | 92 | 96 | **91** | 26.02 | 26.02 | 1.09 | 5.21 |
| 10.0 | 3 | 137 | 136 | 133 | 133 | **130** | 5.11 | 4.41 | 2.26 | 2.26 |
| | 5 | 137 | 135 | 130 | 130 | **126** | 8.03 | 6.67 | 3.08 | 3.08 |
| | 7 | 137 | 135 | 130 | 130 | **126** | 8.03 | 6.67 | 3.08 | 3.08 |

in Figure 10. It uses RPD to stipulate the amount of improvement with the proposed model.

Figure 11 demonstrates the comparison D-GWO with the state-of-the-arts in terms of *speedup*. In all scenarios, D-GWO outperforms others. This figure shows the same results as the previous figures. Table 9 elaborates this comparison in detail.

Figure 12 shows the comparison of D-GWO with the state-of-the-arts in terms of *efficiency*. In all scenarios, D-GWO outperforms others.

Table 10 elaborates this comparison in detail. As Table 10 indicates, D-GWO beats others in terms of *efficiency*, which means that it exploits the underlying infrastructure with the maximum utilization. The MPQGA, D-PSO, C-SA, and PEFT algorithms are placed from the second to the fifth best in terms of system utilization.

In all 12 scenarios, the C-SA competes with PEFT, because the C-SA has local optimization trend and PEFT is only a heuristic domain-specific algo-

**Table 8.** Comparison of the literature in term of *SLR* metric.

| CCR | No. of VMs | *SLR* | | | | | RPD (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | PEFT | C-SA | MPQGA | D-PSO | D-GWO | PEFT | C-SA | MPQGA | D-PSO |
| 0.4 | 3 | 1.99 | 1.94 | 1.89 | 1.86 | **1.86** | 6.53 | 4.12 | 1.59 | 0 |
| | 5 | 1.69 | 1.69 | 1.67 | 1.65 | **1.63** | 3.55 | 3.55 | 2.40 | 1.21 |
| | 7 | 1.65 | 1.62 | 1.58 | 1.58 | **1.55** | 6.06 | 4.32 | 1.90 | 1.90 |
| 1.0 | 3 | 2.79 | 2.71 | 2.32 | 2.32 | **2.26** | 19.00 | 16.61 | 2.59 | 2.59 |
| | 5 | 2.45 | 2.45 | 2.16 | 2.26 | **2.13** | 13.06 | 13.06 | 1.39 | 5.75 |
| | 7 | 2.37 | 2.37 | 2.16 | 2.26 | **2.13** | 10.13 | 10.13 | 1.39 | 5.75 |
| 5.0 | 3 | 5.23 | 5.04 | 3.77 | 3.85 | **3.46** | 33.84 | 31.35 | 8.22 | 10.13 |
| | 5 | 4.73 | 4.92 | 3.69 | 3.69 | **3.65** | 22.83 | 25.81 | 1.08 | 1.08 |
| | 7 | 4.73 | 4.73 | 3.54 | 3.69 | **3.50** | 26.00 | 26.00 | 1.13 | 5.15 |
| 10.0 | 3 | 6.23 | 6.18 | 6.05 | 6.05 | **5.91** | 5.14 | 4.37 | 2.31 | 2.31 |
| | 5 | 6.23 | 6.14 | 5.91 | 5.91 | **5.73** | 8.03 | 6.68 | 3.05 | 3.05 |
| | 7 | 6.23 | 6.14 | 5.91 | 5.91 | **5.73** | 8.03 | 6.68 | 3.05 | 3.05 |

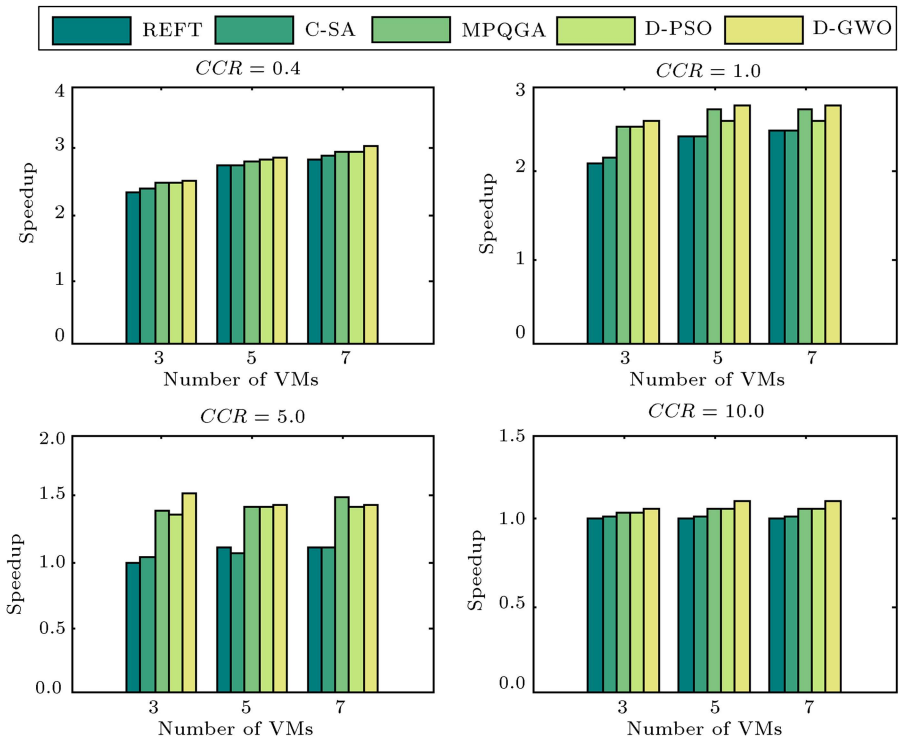


**Figure 11.** Performance comparison of D-GWO with others in terms of *speedup*.

rithm. For this reason, both of them search in a limited region. The C-SA marginally outperforms the PEFT in 7 scenarios out of 12, they are the same in 4 scenarios, and PEFT works better in only one scenario in rather communication-intensive graphs. On the other hand, the main competition is between MPQGA and D-GWO. D-PSO competes with MPQGA in some cases, but in the majority of cases, MPQGA works better. As mentioned earlier, they work the same in 7 scenarios and in one scenario D-PSO works better than MPQGA, but in the remaining 4 scenarios, the MPQGA performs better than D-PSO. The D-GWO beats MPQGA in terms of all evaluation parameters, because it engages pertinent operators and carefully balances exploration

**Table 9.** Comparison of the literature in term of *speedup*.

| CCR | No. of VMs | *Speedup* | | | | | RPD (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | PEFT | C-SA | MPQGA | D-PSO | D-GWO | PEFT | C-SA | MPQGA | D-PSO |
| 0.4 | 3 | 2.34 | 2.40 | 2.47 | 2.47 | **2.51** | 6.97 | 4.48 | 1.49 | 1.49 |
| | 5 | 2.75 | 2.75 | 2.80 | 2.83 | **2.86** | 3.98 | 3.98 | 2.27 | 1.14 |
| | 7 | 2.83 | 2.88 | 2.95 | 2.95 | **3.02** | 6.59 | 4.79 | 2.40 | 2.40 |
| 1.0 | 3 | 2.08 | 2.14 | 2.50 | 2.50 | **2.56** | 23.26 | 19.77 | 2.33 | 2.33 |
| | 5 | 2.37 | 2.37 | 2.68 | 2.56 | **2.72** | 14.81 | 14.81 | 1.23 | 6.17 |
| | 7 | 2.44 | 2.44 | 2.68 | 2.56 | **2.72** | 11.11 | 11.11 | 1.23 | 6.17 |
| 5.0 | 3 | 1.00 | 1.04 | 1.39 | 1.36 | **1.51** | 51.11 | 45.56 | 8.89 | 11.11 |
| | 5 | 1.11 | 1.06 | 1.42 | 1.42 | **1.43** | 29.47 | 34.74 | 1.05 | 1.05 |
| | 7 | 1.11 | 1.11 | 1.48 | 1.42 | **1.43** | 35.16 | 35.16 | 1.10 | 5.49 |
| 10.0 | 3 | 1.00 | 1.01 | 1.03 | 1.03 | **1.05** | 5.38 | 4.6 | 2.31 | 2.31 |
| | 5 | 1.00 | 1.01 | 1.05 | 1.05 | **1.09** | 8.73 | 7.14 | 3.17 | 3.17 |
| | 7 | 1.00 | 1.01 | 1.05 | 1.05 | **1.09** | 8.73 | 7.14 | 3.17 | 3.17 |

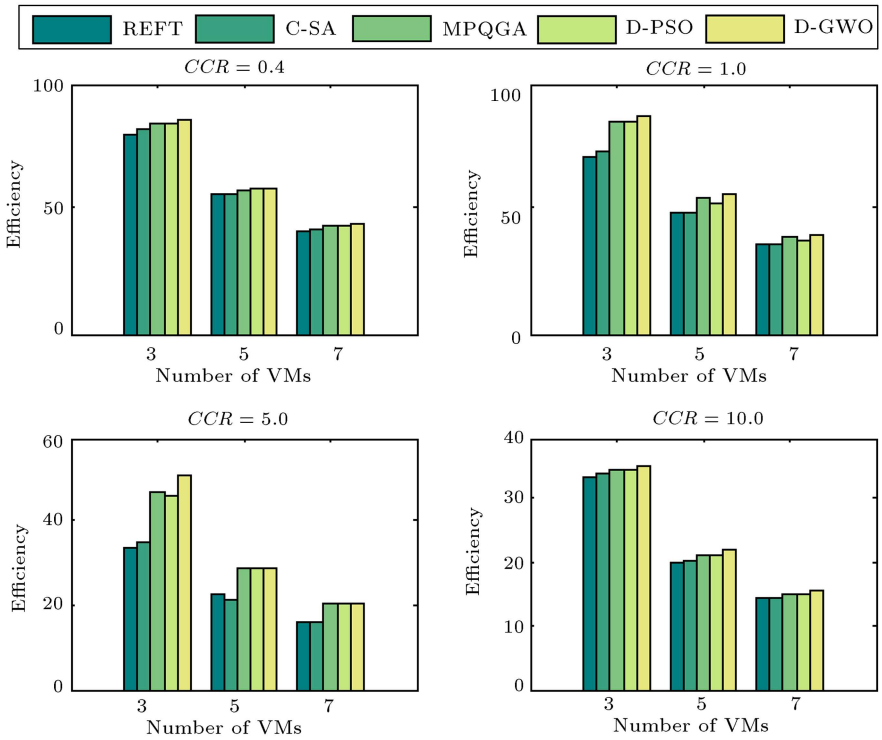


**Figure 12.** Performance comparison of D-GWO with others in terms of *efficiency*.

with the exploitation phase. Another important point is that the improvement inclination is increased by increasing the $CCR$ value, except for $CCR = 10.0$. The reason is that parallel algorithms do not have brilliant improvement in communication-intensive graphs in comparison with serial executions. To prove the scalability of D-GWO, different datasets for larger graphs up to 150 nodes were generated. The results also proved the significant improvement.

*7.3.2. Time complexity*

Algorithm 1 as the main one has several sub-algorithms. Algorithm 2 spends $O(m+n)$ because making each ranking list takes $O(n)$ along with loops taking $O(m)$. Algorithm 3 takes $O(AM)$ where $A$ and $M$ indicate the number of arcs and VMs.

**Table 10.** Comparison of the literature in term of *efficiency*.

| CCR | No. of VMs | *Efficiency (%)* | | | | | RPD (%) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **PEFT** | **C-SA** | **MPQGA** | **D-PSO** | **D-GWO** | **PEFT** | **C-SA** | **MPQGA** | **D-PSO** |
| 0.4 | 3 | 78.00 | 80.00 | 82.33 | 82.33 | **83.79** | 7.27 | 4.58 | 1.62 | 1.62 |
| | 5 | 55.00 | 55.00 | 56.00 | 56.60 | **57.20** | 4.00 | 4.00 | 2.14 | 1.06 |
| | 7 | 40.43 | 41.14 | 42.14 | 42.14 | **43.14** | 6.71 | 4.86 | 2.37 | 2.37 |
| 1.0 | 3 | 69.33 | 71.33 | 83.33 | 83.33 | **85.33** | 23.08 | 19.63 | 2.40 | 2.40 |
| | 5 | 47.40 | 47.40 | 53.60 | 51.20 | **54.40** | 14.77 | 14.77 | 1.49 | 6.25 |
| | 7 | 34.86 | 34.86 | 38.29 | 36.57 | **38.86** | 11.48 | 11.48 | 1.49 | 6.25 |
| 5.0 | 3 | 33.33 | 34.67 | 46.33 | 45.33 | **50.33** | 51.00 | 45.19 | 8.63 | 11.03 |
| | 5 | 22.20 | 21.20 | 28.40 | 28.40 | **28.60** | 28.83 | 34.91 | 0.70 | 0.70 |
| | 7 | 15.86 | 15.86 | 20.14 | 20.29 | **20.43** | 28.83 | 28.83 | 0.70 | 0.70 |
| 10.0 | 3 | 33.33 | 33.67 | 34.34 | 34.34 | **35.00** | 5.00 | 3.96 | 1.94 | 1.94 |
| | 5 | 20.00 | 20.20 | 21.00 | 21.00 | **21.80** | 9.00 | 7.92 | 3.81 | 3.81 |
| | 7 | 14.29 | 14.43 | 15.00 | 15.00 | **15.57** | 9.00 | 7.92 | 3.81 | 3.81 |

Algorithm 4 takes $O(n)$ because of the *for*-loop. Time complexity for all *Walking Around* approaches takes the same $O(n)$, because the permutation takes at most $n$ operations. Finally, Algorithm 1 takes $O(n.m.MaxIteration + A.M)$.

## 8. Conclusion and future work

This paper presented a novel Discrete Grey Wolf Optimizer (D-GWO) to improve *makespan* of workflows running on heterogeneous cloud platforms. To this aim, novel binary vectors and operators were introduced to efficiently explore the discrete search space by making a balance between local and global searches. To cover exploitation, several permutation procedures were designed to enhance the gained solution from exploration. The performance superiority of the proposed method was verified in different circumstances against other state-of-the-arts. Since the users trust reliable computing resources, we contemplate to model cloud reliability for workflow scheduling problems in future work.

## References


1. Hosseini Shirvani, M.S. "A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems", *Eng. Appl. Artif. Intell.*, **90**(December 2019), p. 103501 (Apr. 2020 DOI: 10.1016/j.engappai.2020.103501
2. Armbrust, M., Fox, A., Griffith, A.R., et al., *Above the Clouds: A Berkeley View of Cloud Computing*, University of California, Berkeley (2009).
3. Jafari Navimipour, N. "A formal approach for the specification and verification of a trustworthy human resource discovery mechanism in the expert cloud", *Expert Systems with Applications*, **42**(15-16), pp. 6112–6131 (2015).
4. Roy, S.K., Devaraj, R., Sarkar, A., et al. "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems", *Journal of Systems Architecture*, **105**(101706), pp. 1–26 (2020). DOI: https://doi.org/10.1016/j.sysarc.2019.10170 6
5. Keshanchi, B. and Jafari Navimipour, N. "Priority-based task scheduling algorithm in cloud systems using a memetic algorithm", *Journal of Circuits, Systems, and Computers*, **25**(10), pp. 1–33 (2016).
6. Tong, Z., Chen, H., Deng, X., et al. "A scheduling scheme in the cloud computing environment using deep Q-learning", *Information Sciences*, **512**, pp. 1170–1191 (2020). DOI: https://doi.org/10.1016/j.ins.2019.10.035.
7. Mohammadzadeh, A., Masdari, M., and Gharehchopogh, F.S. "Energy and cost-aware workflow scheduling in cloud computing data centers using a multi-objective optimization algorithm", *J Netw Syst Manage*, **29**(31), pp. 1–34 (2021). https://doi.org/10.1007/s10922-021-09599-4.
8. Hosseini Shirvani, M.S. "A new shuffled genetic-based task scheduling algorithm in heterogeneous distributed systems", *Heterog. Distrib. Syst. J. Adv. Comput. Res.*, **9**(4), pp. 19–36 (2018).

9. Hosseini Shirvani, M.S. "Evaluating of feasible solutions on parallel scheduling tasks with DEA decision maker", *J. Adv. Comput. Res.*, **6**, pp. 109–115 (2015).
10. Topcuoglu, H., Hariri, S., and Wu, M.Y. "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Trans. Parallel Distrib. Syst.*, **13**(3), pp. 260–274 (2002). DOI: 10. 1109/71.993206
11. Arabnejad, H. and Barbosa, J.G. "List scheduling algorithm for heterogeneous systems by an optimistic cost table", *IEEE Trans. Parallel Distrib. Syst.*, **25**(3), pp. 682–694 (Mar. 2014). DOI: 10.1109/TPDS.2013. 57
12. Thaman, J. and Singh, M. "Green cloud environment by using robust planning algorithm", *Egypt. Informatics J.*, **18**(3), pp. 205–214 (Nov. 2017). DOI: 10.1016/j.eij.2017.02. 001
13. Khan, M. "Scheduling for heterogeneous Systems using constrained critical paths", *Parallel Comput.*, **38**, pp. 175–193 (2012). DOI: 10.1016/j.parco.2012.01.001
14. Lin, C.-S., Lin, C.-S., Lin, Y., et al. "Multi-objective exploitation of pipeline parallelism using clustering, replication and duplication in embedded multi-core systems", *J. Syst. Archit.*, **59**(10), pp. 1083–1094 (Nov. 2013). DOI: 10.1016/j.sysarc.2013.05.024
15. Liou, J. and Palis, M.A. "An efficient task clustering heuristic for scheduling DAGs on multiprocessors", *Symp. Parallel Distrib. Process.* February, pp. 152–156 (1996).
16. Tang, Q., Zhu, L.-H., Zhou, L., et al. "Scheduling directed acyclic graphs with optimal duplication strategy on homogeneous multiprocessor systems", *J. Parallel Distrib. Comput.*, **138**, pp. 115–127 (Apr. 2020) DOI: 10. 1016/j.jpdc.2019.12.012.
17. Akbari, M., Rashidi, H., Alizadeh, S.H. "An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems", *Eng. Appl. Artif. Intell*, **61**, pp. 35–46 (2017). http://dx.doi.org/10.1016/j.engappai.2017.02.013.
18. Zand, H.V., Raji, M., Pedram, H., et al. "A genetic algorithm-based tasks scheduling in multicore processors considering energy consumption", *International Journal of Embedded Systems (IJES)*, **13**(3), pp. 264–273 (2020).
19. Sujana, J.A.J., Revathi, T., Priya, T.S.S., et al. "Smart PSO-based secured scheduling approaches for scientific workflows in cloud computing", *Soft Comput.*, **23**, pp. 1745–1765 (2019). https://doi.org/ 10.1007/s00500-017-2897-8
20. Dordaie, N. and Jafari Navimipour, N., *A Hybrid Particle Swarm Optimization and Hill Climbing Algorithm for Task Scheduling in the Cloud Environments*, ICT Press., **4**(4), pp. 199–202 (Dec. 2018).
21. Alsaidy, S.A., Abbood, A.D., and Sahib, M.A. "Heuristic initialization of PSO task scheduling algorithm in cloud computing", *J. King Saud Univ. – Comput. Inf. Sci.*, **34**(6A), pp. 2370–2382 (2022). https://DOI.org/10.1016/ j.jksuci.2020.11.002
22. Boveiri, H.R. "An enhanced cuckoo optimization algorithm for task graph scheduling in cluster-computing systems", *Soft Comput* , **24**, pp. 10075–10093 (2020). https://DOI.org/10.1007/s00500-019-04520-3
23. Agrawal, M. and Sirvastava, G.M.S. "A cuckoo search algorithm-based task scheduling in cloud computing", In *Advances in Computer and Computational Sciences* (2018). DOI: 10.1007/978-981-10-3773-3_29
24. Moschakis, I.A. and Karatza, H.D. "Multi-criteria scheduling of bag-of-tasks applications on heterogeneous interlinked clouds with simulated annealing", *The Journal of Systems & Software*, **101**, pp. 1–14 (2015). DOI: 10.1016/j.jss.2014.11.014
25. Osamy W., El-sawy A.A., and Khedr, A.M., "SATC: A simulated annealing based tree construction and scheduling algorithm for minimizing aggregation time in wireless sensor networks", *Wireless Pers Commun*, **108**, pp. 921–938 (2019), https://doi.org/10.1007/s11277-019-06440-9
26. De Vicente, J., Lanchares, J., and Hermida, R. "Placement by thermodynamic simulated annealing", *A Physics Letters*, **317**(56), pp. 415–423 (2013).
27. Keshani, M. and Jahanshahi, M.H. "Using simulated annealing for task scheduling in distributed systems", *International Conference on Computational Intelligence, Modelling and Simulation* (2009).
28. Hosseini Shirvani, M.S. and Noorian Talouki, R. "Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach", *Complex Intell. Syst.*, **8**, pp. 1058–1114 (2022). https://doi.org/10.1007/s40747-021-00528-1
29. Mirjalili, S., Mirjalili, S.M., and Lewis, A. "Grey wolf optimizer", *Adv. Eng. Softw.*, **69**, pp. 46–61 (2014).
30. Gu, J., Jiang, T., Zhu, H., et al. "Low-carbon job shop scheduling problem with discrete genetic-grey wolf optimization algorithm", *Journal of Advanced Manufacturing Systems*, **19**(1), pp. 1–14 (2020). doi: 10.1142/S0219686720500018
31. Mohammadzadeh, A., Masdari, M., Gharehchopogh, F.S., et al. "Improved chaotic binary grey wolf optimization algorithm for workflow scheduling in green cloud computing", *Evol. Intel.*, **14**, pp. 1997–2025 (2021). https://doi.org/10.1007/s12065-020-00479-5
32. Hosseini Shirvani, M.S., Amirsoleimani, N., Salimpour, S., and Azab, A. "Multi-criteria task scheduling in distributed systems based on fuzzy TOPSIS," *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4 (2017). DOI: 10.1109/CCECE.2017.7946721
33. Guo, P. and Xue, Z. "Cost-effective fault-tolerant scheduling algorithm for real-time tasks in cloud systems", *17th IEEE International Conference on Communication Technology* (2017).
34. Noorian Talouki, R., Hosseini Shirvani, M.S., and Motameni, H. "A heuristic-based task scheduling algorithm for scientific workflows in heterogeneous cloud computing platforms", *Journal of*

*King Saude University: Computer and Information Sciences*, **34**(8A), pp. 4902–4913 (2022). https://doi.org/10.1016/j.jksuci.2021.05.011.

35. Darbha, S. and Agrawal, D.P. "A task duplication based scalable scheduling algorithm for distributed memory systems", *Journal of Parallel and Distributed Computing*, **46**, pp. 15–27 (1997).
36. Palis, M.A., Liou, J.C., and Wie, D.S.L. "Task clustering and scheduling for distributed memory parallel architectures", *IEEE Transactions on Parallel and Distributed Systems*, **7**(1), pp. 46–55 (1996).
37. Xu, Y., Li, K., Hu, J., et al. "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues", *Information Sciences*, **270**, pp. 255–287 (2014).
38. Mohammadzadeh, A., Masdari, M., Gharehchopogh, F.S., et al. "A hybrid multi-objective metaheuristic optimization algorithm for scientific workflow scheduling", *Cluster Comput*, **24**, pp. 1479–1503 (2021). https://doi.org/10.1007/s10586-020-03205-z.
39. Noorian Talouki, R., Hosseini Shirvani, M.S., and Motameni, H. "A hybrid meta-heuristic scheduler algorithm for optimization of workflow scheduling in cloud heterogeneous computing environment", *Journal of Engineering, Design and Technology* (2021). https://doi.org/10.1108/JEDT-11-2020-0474.
40. Tanha, M., Hosseini Shirvani, M.S., and Rahmani A.M. "A hybrid meta-heuristic task scheduling algorithm based on genetic and thermodynamic annealing algorithms in cloud computing environemnts", *Neural Computing and Applications*, **33**, pp. 16951–16984 (2021). https://doi.org/10.1007/s00521-021-06289-9
41. Javadian Kootanaee, A., Poor Aghajan, A., and Hosseini Shirvani, M.S. "A hybrid model based on machine learning and genetic algorithm for detecting fraud in financial statements", *Journal of Optimization in Industrial Engineering*, **14**(2), pp. 169–186 (2021). doi:10.22094/joie.2020.1877455.1685
42. Bharathi, S., Chervenak, A., Deelman, E., et al. "Characterization of scientific workflows", In *Third Workshop on Workflows in Support of Large-Scale Science*, pp. 1–10 (2008). DOI: 10.1109/WORKS.2008.4723958.

## Biography

**Mirsaeid Hosseini Shirvani** received his BSc, MSc, and PhD degrees all in Computer Software Engineering Systems from Universities in Tehran, Iran. He has been teaching miscellaneous computer courses in several universities in Mazandaran province since 2001. He has also published several papers in authentic and worldwide well-reputed journals. Currently, he is a professor in the Computer Engineering Department at IAU (serving as a faculty member of the Sari-Branch). His research interests are in the areas of cloud- and fog-computing, IoT, distributed systems, parallel processing, machine learning, and evolutionary computations.