A Novel Discrete Grey Wolf Optimizer for Scientific Workflow Scheduling in Heterogeneous Cloud Computing Platforms

Mirsaeid Hosseini Shirvani

1) Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran

Corresponding Author: mirsaeid_hosseini@iausari.ac.ir & mirsaeid_hosseini@yahoo.com

Corresponding author address: Mirsaeid Hosseini Shirvani (Cell Number: +989111144575), Faculty of Computer Engineering, Islamic Azad University, Sari, Iran, Post Code: 48161 – 19318.

Abstract

There are several scientific workflow applications which need vast amount of processing so the cloud offerings give the sense of economic for them. Workflow scheduling has drastic impact on gaining desired quality of service (QoS). The main objective of workflow scheduling is to minimize the makespan. This scheduling is formulated to 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 in the aim of minimizing makespan. Although traditional grey wolf optimizer (GWO) had great achievements in continuous optimization problems, it seems clear gap in utilizing GWO for combinatorial discrete optimization problems. It revolves around the fact that the continuous changes in search space during the course of discrete optimization lead inefficient or meaningless solutions. To this end, the proposed algorithm is customized to optimize discrete workflow scheduling problem by leveraging some new binary operators and Walking Around approaches to balance between exploration and exploitation in discrete search space. Scientific unstructured workflows were investigated in different circumstances to prove the effectiveness of proposed D-GWO. The simulation results witnessed the superiority of proposed D-GWO against other state-of-the-arts in terms of scheduling assessment metrics.

Keyword: cloud computing; scientific workflow scheduling; meta-heuristic algorithm; discrete grey wolf optimization; walking around technique;

1. Introduction

Cloud computing presents itself as utility computing to their 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 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 different performance. Thus, exploited different workflow scheduling approaches lead different outcomes. The workflow scheduling which determines what task should be assigned on what VM for execution is NP-hard [5-6]. It is impossible to find optimal scheduling in the bounded time frame. To address the issue, many heuristic and meta-heuristic algorithms were published to solve scheduling problems. In this domain, the most important quality of service (QoS) parameter is makespan or total execution time [7-8]. So, the main concentration of scheduling proposals is on makespan improvement [9]. In scheduling context, one of the earliest algorithms was known as list schedulers [10]. Each list scheduler firstly produces a list including ordered tasks associated to a given workflow. This list must guarantee to be a topological sort not to violate the precedence constraints. It secondly assigns the high priority unscheduled task on 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 for its first step [10]. Some extensions of HEFT were improved that are predictable earliest finish time (PEFT) [11], robust earliest finish time (RHEFT) [12], and constrained earliest finish time (CEFT) [13]. Moreover, variety heuristics were 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 couple of processors to increase parallelism. In task clustering technique, some tasks are ground in a cluster to be executed on the same VM [14]. By utilizing this, the data transfer time is omitted in which it can potentially decrease makespan. At last, data replication method uses data pipeline to reduce VMs’ idle time [16]. In the larger problems, the majority of search space remains unexplored by utilizing either heuristics or list schedulers. Therefore, miscellaneous meta-heuristics were proposed to improve the scheduling quality which mainly are based on genetic algorithm (GA) [5, 8, 17, 30], particle swarm optimization (PSO) [1, 18-19, 29], cuckoo search optimization (CSO) [20-21], and simulated annealing (SA) [22-25, 34]. One of the most successful meta-heuristic algorithms which solves continuous optimization problems is grey wolf optimization (GWO) [40-41]. The traditional GWO cannot efficiently solve discrete problems [35]. It revolves around the fact that the continuous changes and modifications in search space during the course of discrete optimization lead inefficient or meaningless solutions. The majority of meta-heuristics have global trend in their exploration phase in which they neglect exploitation of current solution or balance between them. In this paper, a novel discrete GWO is presented to solve combinatorial workflow scheduling problem in cloud environment with heterogeneous platform. To this end, new binary operators and Walking Around techniques with couple of procedures are proposed to balance between exploration and exploitation of 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 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 attack process
The reset of the paper is organized as follows. Section 2 presents related works. A review on original GWO concepts is brought in section 3. Section 4 provides proposed models and problem formulation. Section 5 brings an illustrative example. Section 6 is dedicated to proposed novel discrete grey wolf optimizer for workflow scheduling problem. Performance evaluation of the proposal is placed in section 7. Section 8 includes conclusion and future direction.

2. Related Works

The review on scheduling algorithms helps to categorize proposals 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 high priority task to a VM that returns 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] were proposed in which they take different QoS criteria [27]. The cost-effective fault tolerant workflow scheduling was suggested for execution of real time applications on cloud datacenter which encounters transient and permanent failures [31]. Variety heuristics are added to improve the performance of workflow schedulers. Duplication and clustering methods are two important approaches [28, 32-33]. The task duplication copies critical tasks of a given workflow on couple of VMs to increase the degree of parallelism, but the clustering method groups some highly dependent tasks in a cluster; then, all tasks in the cluster are assigned on the same VM. It potentially reduces 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 for small scale problems or in limited time window for quick decision. Therefore, several meta-heuristics were extended to solve workflow scheduling problems. A shuffle-based genetic algorithm was customized to solve workflow scheduling in distributed systems [8]. The similar algorithm applied multi-queue besides GA to produce semi random initialization [26]. A scheduling algorithm that engages PSO was proposed which suffers from early convergence [18]. The literature review is summarized in Table 1.

The review reveals that the majority of proposals suffer from balancing local and global searches during the course of optimization. In addition, the potential improvement remained in exploring discrete search space. To fill the gaps, current paper intends to bridge the 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 kinds of wolves; the first, second, and third bests are alpha (α), beta (β), and delta (δ) respectively; the rest wolves are omega (ω). For predating behavior, each wolf is conducted by three kinds of wolves α, β, and δ. In their predatory behavior, three main stages are performed, namely, encircling the prey, hunting, and attacking the prey. The first two are utilized for exploration whereas the third stage is engaged for exploitation in search space. For encircling the prey, each individual X(t) in round t-th of
optimization course finds its distance to a guessed prey \( X_p(t) \); then, it adjusts its path towards the prey. The distance value and adjust of direction are calculated via Eq. (1) and Eq. (2).

\[
\overline{D} = \left| \overline{C}.X_p(t) - \overline{X}(t) \right|
\]

(1)

\[
\overline{X}(t+1) = \overline{X}_p(t) - \overline{A.D}
\]

(2)

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

\[
\overline{A} = 2\overline{a}.r_1 - \overline{a}
\]

(3)

\[
\overline{C} = 2r_2
\]

(4)

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

\[
\overline{X}_a = [\overline{C}_1.\overline{X}_a - \overline{X}], \overline{X}_\beta = [\overline{C}_2.\overline{X}_\beta - \overline{X}], \overline{X}_\delta = [\overline{C}_3.\overline{X}_\delta - \overline{X}]
\]

(5)

\[
\overline{X}_1 = \overline{X}_a - \overline{A}_1.\left(D_a\right), \overline{X}_2 = \overline{X}_\beta - \overline{A}_2.\left(D_\beta\right), \overline{X}_3 = \overline{X}_\delta - \overline{A}_3.\left(D_\delta\right)
\]

(6)

\[
\overline{X}(t+1) = \frac{\overline{X}_1 + \overline{X}_2 + \overline{X}_3}{3}
\]

(7)

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

### 4. Models and Problem Statement

Several models are presented to be ready for problem definition.

#### 4.1 System and Application Models
The cornerstone of a cloud system is a datacenter. Each datacenter provides a list of heterogeneous virtual machines \( VMlist = \{ VM_1, VM_2, \ldots, VM_q \} \). Each VM is determined in term of variable million instruction per second (MIPS). Fig. 1 illustrates the proposed system model for cloud environment.

The users request workflow execution which is received via cloud’s front end module. Then the scheduler, embedded in the cloud broker, assigns tasks on available VMs to meet user’s requirement. Workflow applications are modeled in DAGs. Each DAG contains nodes and arcs. A workflow \( W \) contains \( n \) tasks \( W = \{ t_1, t_2, \ldots, t_n \} \) and set of arcs \( A = \{ (t_i, t_j) \mid t_i, t_j \in W \} \). 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 tasks. 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).

\[
ET(t_i, VM_j) = \frac{(MIS)\text{assigned} - to - a - task - t_i}{(MIPS)\text{assigned} - to - a - processor - VM_j}
\]  

(8)

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

\[
\frac{ET(t_i)}{q} = \left( \sum_{j=1}^{q} ET(t_i, VM_j) \right)
\]  

(9)

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

\[
C(t_i, t_j) = L + \frac{DV}{BW}
\]  

(10)

The term \( L \) is relevant to intrinsic link’s delay which is negligible and term \( DV \) is the amount of data volume transferring on network bandwidth (BW). If schedulers assign two dependent tasks on the same VM, the communication cost is omitted. Fig. 2 depicts a DAG where \( t_1 \) and \( t_{10} \) are entry and exit nodes respectively.

Table 2 presents execution time for each task once it executes on each of 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 domain is to use the communication-to-computation rate (CCR) concept calculated by Eq. (11).

\[
CCR = \frac{1}{|A|} \left( \sum_{\text{edge}(i,j)} C(t_i,t_j) \right) - \frac{1}{|W|} \left( \sum_{t_i} ET(t_i) \right)
\]

(11)

If the CCR value is high the given workflow is relatively communication–intensive. The value of CCR for a DAG of Fig. 2 is 0.83 that means it is a moderate graph.

4.2 Scheduling Model and Problem Formulation

Task scheduling of the workflow execution is a very important mission because it determines what available VM should be assigned to which task. The proposed scheduling model follows two steps: to prioritize tasks and to select VM for assignment to tasks [10]. To prioritize tasks, three ranking procedures are engaged that are upward, downward, and level ranking procedures; 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 \(\text{Succ}(t_i)\) and \(\text{Pred}(t_i)\) 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 exit node by Eq. (12), but for other nodes the ranking values are calculated by incorporation of Eq. (13).

\[
\text{UpRank}(t_{\text{exit}}) = ET(t_{\text{exit}})
\]

(12)

\[
\text{UpRank}(t_i) = ET(t_i) + \max_{t_j \in \text{succ}(t_i)} \{\text{UpRank}(t_j) + C(t_j,t_i)\}
\]

(13)

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

\[
\text{DownRank}(t_{\text{entry}}) = 0
\]

(14)

\[
\text{DownRank}(t_i) = \max_{t_j \in \text{pred}(t_i)} \{\text{DownRank}(t_j) + ET(t_j) + C(t_j,t_i)\}
\]

(15)

Finally, the level ranking procedure starts from entry to reach exit. It calculates level ranking value as zero for entry, but for others these are calculated by Eq. (16).

\[
\text{LevelRank}(t_i) = \max_{t_j \in \text{pred}(t_i)} \{\text{LevelRank}(t_j)\} + 1
\]

(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, functions Earliest Finish Time (EFT) and Earliest Start Time (EST) are exploited. The function \( EFT(t_i, VM_j) \) 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 time which the execution of \( t_i \) can be begun on virtual machine \( VM_j \). This function considers both \( VM_j \)'s availability time and receiving the data prerequisites of the given task \( t_i \) into account.

For entry and non-entry tasks, \( EST(t_i, VM_j) \) is calculated by Eq. (17) and Eq. (18).

\[
EST(t_{entry}, VM_j) = 0
\]

\[
EST(t_i, VM_j) = \max \left\{ \text{Avail}(VM_j), \max_{\nu_{\nu_{\text{pred}}(t_i)}} \left\{ AFT(t_q) + C(t_q, t_i) \right\} \right\}
\]

In Eq. (18), the maximum value between \( \text{Avail}(VM_j) \) and the latest time that the prerequisite data of \( t_i \) is received must be considered. The term \( \text{Avail}(VM_j) \) indicates the earliest time that VM is free to do 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 earliest finish time. The term \( \text{index} \) indicates to the number of VM in the \( VMList \) that returns the minimum value.

\[
AFT(t_q, VM_{\text{index}}) = \min_{\nu_{\nu_{VM, vMList}}} \left\{ EFT(t_q, VM_j) \right\}
\]

In addition, the \( EFT(t_i, VM_j) \) is calculated by summation of two values \( EST(t_i, VM_j) \) and \( ET(t_i, VM_j) \) that Eq. (20) draws.

\[
EFT(t_i, VM_j) = EST(t_i, VM_j) + ET(t_i, VM_j)
\]

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

\[
\text{makespan} = \min \left\{ \max_{\nu_{\nu_{\text{w}}}} \left( AFT(t_i) \right) \right\}
\]

Since existing schedulers present limited number of valid tasks lists, there is a clear lack to find optimal solution with efficiently exploring search space. Therefore, D-GWO is extended to bridge the gap.

5. An Illustrative Example

An illustrative example shows the effectiveness of D-GWO in workflow scheduling. Fig. 2 is considered as a case study. D-GWO is compared against other state-of-the-arts. The comparatives are two famous list schedulers \( HEFT \) [10] and \( PEFT \) [11]; two meta-heuristics: multiple priority queues and genetic algorithm (MPQGA) [5] and a customized simulated annealing-based (C-SA) [23]; and a hybrid D-PSO [42]. Table 3 returns the rank values of each
task derived by each algorithm. Table 4 shows lists of tasks generated by different approaches along with gained final makespan. Fig. 3 illustrates the performance of D-GWO against other literatures; it proves D-GWO dominates others in term of makespan.

6. Proposed Discrete Grey Wolf Optimization Algorithm for Solving Workflow Scheduling Problem

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

6.1. Problem Encoding (Memetic & Wolf Representation)

The problem encoding phase is one of the most important issues that has impact on tracking and performance of proposed meta-heuristic algorithm [37-39]. The task name is considered as an allele; so, genes are selected from set of integer numbers \{1,2,\ldots,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 \( \text{List}_{D-GWO} = \{t_1, t_4, t_3, t_6, t_7, t_9, t_8, t_{10}\} \) is encoded to a wolf illustrated in Fig. 4.

6.2. Auxiliary Binary Vectors and Operators

The trajectory of individual wolf toward a prey is conducted by three best wolves alpha (\( W_\alpha \)), beta (\( W_\beta \)), and delta (\( W_\delta \)). Therefore, new auxiliary binary vectors and operators are proposed to take benefit of leader wolves’ knowledge about traversed discrete search space. Thus, binary vectors \( \text{Token}_i=(b_1, b_2, \ldots, b_n) \) and \( \text{Adjuster}_i=(a_1, a_2, \ldots, a_n) \) are applied for comparison between each agent and leader wolves. To be conducted, each wolf must relocate tasks in the task list similar to leaders’ encodings. It the zero bit means the corresponding task is not necessary to be changed. Note that, \( t_{\text{entry}} \) and \( t_{\text{exit}} \) are not to be changed; so, they are set to zero in the \text{Token} vector. Furthermore, in the initialization all tasks are exposed to be changed that is why the initial value of \text{Token} vector is initialized one.

Binary operators \( \setminus \) and \( \otimes \):

The operator \( \setminus \) is used to indicate the differences in corresponding tasks of two wolves. For instance, take \( n=6 \), \( \text{Token}_1=(0,1,0,1,0) \) for \( W_1 \), and \( \text{Token}_\alpha=(0,0,1,1,1,0) \) for \( W_\alpha \). Then, \( \text{Token}_1 \setminus \text{Token}_\alpha=(0,1,0,1,0,0) \); it means, the output of the same bit is zero because it is not necessary to be changed. For operator \( \otimes \), the associated bit is changed provided the corresponding \text{Adjuster} value is one. If \( \text{Token}_1=(0,1,1,0,0,0) \) and \( \text{Adjuster}_1=(0,0,1,0,1,0) \), then \( \text{Token}_1 = \text{Token}_1 \otimes \text{Adjuster}_1 = (0,1,1,0,0,0) \otimes (0,0,1,0,1,0) = (0,1,0,0,1,0) \). Adjuster vector is a clue for \text{Token} vector to close off duplicate changes on especial tasks.

8
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 which its corresponding binary $b_k$ is one. This substitution is done by a heuristic. The zero value means no change is required. The corresponding $Adjuster$ value is used to change $Token$ value for 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 presented in Fig. 5 starts with initial population by calling Algorithm 2 that utilizes theorems in [1]; the theorems allow to permute tasks of the same level in a list. The vectors $Token$ and $Adjuster$ are set to $\vec{1}$ which means all tasks are exposed to be changed. Then, each wolf is evaluated by fitness which Algorithm 3 calculates. The first, second, and third bests are known 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 criteria is met. In each iteration, for each wolf some operations are performed. Firstly, Algorithm 4 is called to encircling the prey for exploration. If the changes on a given wolf is valid, the best so far solution can be updated otherwise the wolf’s changes are retreated.

Also, the $Token$ is updated based on $Adjuster$; moreover, the $Adjuster$ is updated in Algorithm 4 to preclude duplicate changes. Afterwards, the exploitation phase is started. The Walking Around can potentially improve the solutions of exploration phase. To this end, a random integer is drawn to call one of the Walking Around procedures casually. In Walking Around process, procedures which make permutation are introduced. These procedures are well defined in such a way that to permute search space efficiently. After the exploitation phase is done and changes happened, all of the wolves are evaluated again to find three new leaders alpha, beta, and delta wolves. Finally, the best so far solution is returned. Algorithm 2 depicted in Fig. 6 generates semi-random initial population. It utilizes theorems of Ref. [1] which produce promising individuals. Algorithm 3 illustrated in Fig. 7 calculates fitness function. Algorithm 4 illustrated in Fig. 8 adopts position information of individual and leader wolves: alpha, beta, and delta. Moreover, their $Token$ and $Adjuster$ vectors are considered as input. Based on leaders’ fitness three probability parameters: $P_1$, $P_2$, and $P_3$ are calculated to get a chance for adopting parts of leader wolves’ knowledge about search space for individual wolf toward encircling the prey. Firstly, difference $Token$ ($TokenDiff$) between $Token$ of $W_i$ and other leaders’ $W_\alpha$, $W_\beta$, and $W_\delta$ are obtained. For each task in the list, it is randomly determined that which part is drawn from which leader wolf. Afterwards, the update trajectory is performed by incorporating $Adjuster$ vector. The value zero used in line #16 means the entry and exit tasks are not to be changed. In line #17, the tasks which their corresponding bit value in $Adjuster$ is equal to one are candidates for exchange. After exchange, the new individual is generated as the output.

Walking Around Procedures (For Exploitation)
Here, Walking Around procedures for a given solution are introduced. Three procedures which permute discrete search space are Permutation\textsubscript{1}, Permutation\textsubscript{2}, and Permutation\textsubscript{3}. Having, one of the Walking Around procedures is randomly called. Fig. 9 presents Permutation\textsubscript{1} procedure along with its application. Algorithm 5 as the first kind of permutation randomly opts a meme W\textsubscript{i}. Then, it quests for finding its first successor task in this list such as W\textsubscript{j}. A selected meme W\textsubscript{i} must be exchanged by W\textsubscript{k} in which k \in [i+1..j-1] so that the last predecessor of meme W\textsubscript{k} is ahead of W\textsubscript{i} in the ordered list. It definitely keeps the topological sorting attribute. Fig. 10 illustrates how the Permutation\textsubscript{1} performs. It randomly draws W\textsubscript{4} = t\textsubscript{2}. The, first successor of t\textsubscript{2} is W\textsubscript{7} = t\textsubscript{9}. The Algorithm 5 finds k \in [5..6] so that W\textsubscript{k}’s last predecessor task is ahead of W\textsubscript{4}. In this case, W\textsubscript{6}’s last predecessor task which is t\textsubscript{1} is ahead of W\textsubscript{4}; the reason why W\textsubscript{6} is selected to be exchanged by W\textsubscript{4}. The new wolf is depicted in Fig. 10. Fig. 11 presents Permutation\textsubscript{2} procedure. Algorithm 6 as the second kind of permutation randomly opts a meme W\textsubscript{i} in a wolf W. Then, it quests for finding its last predecessor task in this list such as W\textsubscript{j}. A selected meme W\textsubscript{i} must be exchanged by W\textsubscript{k} in which k \in [j+1..i-1] so that the first successor of meme W\textsubscript{k} is placed after of W\textsubscript{i} in the ordered list. It definitely keeps the topological sorting attribute. Fig. 12 illustrates how the Permutation\textsubscript{2} acts. It randomly draws W\textsubscript{7} = t\textsubscript{9}. The, last predecessor of t\textsubscript{9} is W\textsubscript{4} = t\textsubscript{2}. The Algorithm 6 finds k \in [5..6] so that W\textsubscript{k}’s first successor task is placed after of W\textsubscript{7}. In this case, W\textsubscript{6}’s first successor task which is t\textsubscript{8} is placed after of W\textsubscript{7}; the reason why W\textsubscript{6} is selected to be exchanged by W\textsubscript{7}. The new wolf is depicted in Fig. 12. Fig. 13 presents Permutation\textsubscript{3} procedure. Algorithm 7 as the third kind of permutation firstly measures DAG’s maximum level. Then, it randomly opts two independent memes that are associated two different levels. The independent tasks are substituted. If the new born wolf is valid, it is definitely returned; otherwise the first wolf without change is returned. Fig. 14 illustrates how Permutation\textsubscript{3} works. Two memes W\textsubscript{5} = t\textsubscript{5} and W\textsubscript{7} = t\textsubscript{7} are independent tasks belonging to two different levels L=2 and L=3 respectively. Calling Permutation\textsubscript{3} improves makespan from 138 to 118 depicted in Fig. 14.

### 7. Performance Evaluation

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

#### 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). Utilizing only makespan does not indicate how the scheduling works efficiently. Therefore, the makespan must be compared against the critical path (CP) of a given DAG. The CP is the longest serial path which is not parallelizable. So, the makespan is usually longer than this length. The reason why the new parameter scheduling length ration (SLR) is introduced that is calculated by Eq. (22).
Another important parameter, is to compute how proposal makes speedup. The speedup value measured by Eq. (23) means the reverse relative parallel execution time against serial execution time.

\[
\text{speedup} = \frac{\text{Serial execution - on - VM}}{\text{makespan}} = \frac{\min_{VM_k \in \text{VMList}} \left( \sum_{t_j \in [T]} W(t_j, VM_k) \right)}{\text{makespan}}
\]

(23)

The speedup does not show how many processors are involved in gaining speedup. The reason why the auxiliary parameter efficiency is introduced by Eq. (24).

\[
\text{efficiency} = \frac{\text{speedup}}{\text{Number - of - Used - VMs}} \times 100\%
\]

(24)

7.2 Dataset

The molecular dynamics depicted in Fig. 15 is one of the most important scientific workflows that is pervasively used in physics branches [10, 17, 36]. It is well suited workflow for testing because of its unstructured and unbalanced shape. To evaluate efficiently, several datasets are produced to generate different workflows with different attributes. So, different molecular workflows are generated for simulation datasets. Table 5 elaborates datasets.

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

7.3 Results and Discussion

This section clarifies simulation settings and discusses the results.

Parameter Settings and Performance Analysis

The proposed D-GWO scheduler is compared with several schedulers existing in literature. To this end, the most efficient ones are selected that are PEFT [11], MPQGA [5], C-SA [23], and D-
PSO [42]. All experiments are executed in fair conditions on the same platform. Table 6 depicts parameter settings of each.

Each scenario was independently executed 20 times; the average results are reported in terms of makespan, SLR, speedup, and efficiency. Fig. 16 illustrates the 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 to stipulate the amount improvement of proposal [1]. As Table 7 shows, after D-GWO as the best, D-PSO competes MPQGA in some cases, but in majority of cases MPQGA works better. In the other words, they work the same in 7 scenarios, in one scenario D-PSO works better, but in the rest 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.

Fig. 17 depicts the comparison of D-GWO against other state-of-the-arts in term of SLR which is a normalized parameter regardless to graph shape and size. Again, in all scenarios, D-GWO outperforms others in term of SLR. After D-GWO as the best, the MPQGA, D-PSO, C-SA, and PEFT are placed from second to fifth best.

Table 8 is dedicated to elaborate Fig. 17’s information. It uses RPD to stipulate the amount improvement of proposal.

Fig. 18 demonstrates the comparison D-GWO against state-of-the-arts in term of speedup. In all scenarios, D-GWO outperforms others. This figure shows the same treat that the previous figures showed. Table 9 elaborates this comparison in details.

Fig. 19 shows the comparison of D-GWO against state-of-the-arts in term of efficiency. In all scenarios, D-GWO outperforms others.

Table 10 elaborates this comparison in details. As Table 10 elaborates, the D-GWO beats others in term of efficiency which means it exploits underlying infrastructure with maximum utilization. The MPQGA, D-PSO, C-SA, and PEFT algorithms are placed from second to fifth best in term 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 which is domain-specific algorithm; for this reason both of them search in limited region. The C-SA marginally outperforms in 7 scenarios out of 12 and are the same in 4 scenarios, and works worse than PEFT only in 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 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 rest 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 between exploration and exploitation phases. Another important is that the improvement inclination is increased by increasing in CCR value except for $CCR=10.0$. Because 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 was generated. The results also proved the significant improvement.

**Time Complexity**

Algorithm 1 as the main calls 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 to number of arcs and VMs. Algorithm 4 takes $O(n)$ because of for-loop. Time complexity of all Walking Around approaches take 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 to improve makespan of workflows running on heterogeneous cloud platforms. To prepare this, new binary vectors and operators were introduced to efficiently explore discrete search space with balancing between local and global searches. To cover exploitation, several permutation procedures were designed to enhanced gained solution from exploration. The performance superiority of proposal was verified in different circumstances against other state-of-the-arts. Since the users trust on reliable computing resources, we contemplate to model cloud reliability for workflow scheduling problems in future work.

**References**


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

Fig. 1. The proposed System model

Fig. 2. A typical workflow [11].

Fig. 3.a) D-GWO scheduler, makespan=118

Fig. 3.b) PEFT scheduler [11], makespan=122
Fig. 3.c) MPQGA scheduler [5], \(\text{makespan}=122\)

Fig. 3.d) S-CA scheduler [23], \(\text{makespan}=124\)

Fig. 3.e) HEFT-upward scheduler [10], \(\text{makespan}=133\)

Fig. 3.f) D-PSO scheduler [42], \(\text{makespan}=122\)

Fig. 3. An Illustrative Example

Fig. 4. An encoded wolf

Fig. 5. Proposed D-GWO

Fig. 6. Generating initial wolves

Fig. 7. Fitness calculation

Fig. 8. Encircle the prey

Fig. 9. Walking Around \(\text{Permutation}_1\)

Fig. 10. Performance of \(\text{Permutation}_1\)

Fig. 11. Walking Around \(\text{Permutation}_2\)

Fig. 12. Performance of \(\text{Permutation}_2\)

Fig. 13. Walking Around \(\text{Permutation}_3\)

Fig. 14. Performance of \(\text{Permutation}_3\)

Fig. 15. Molecular workflow [10, 17, 36].

Fig. 16. Performance comparison D-GWO against others in term of \(\text{makespan}\)

Fig. 17. Performance comparison D-GWO versus others in term of \(\text{SLR}\)

Fig. 18. Performance comparison D-GWO versus others in term of \(\text{speedup}\)

Fig. 19. Performance comparison D-GWO versus others in term of \(\text{efficiency}\).

Table 1. The summary of literature

Table 2. Execution time on VMs [11].

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

Table 4. Ordered list of tasks produced by comparative algorithms along with final \(\text{makespan}\)

Table 5. Different simulation datasets.

Table 6. Parameter Settings of Comparative Algorithms.

Table 7. Comparison of literatures in term of \(\text{makespan}\).

Table 8. Comparison of literature in term of \(\text{SLR}\) metric.

Table 9. Comparison of literature in term of \(\text{speedup}\).

Table 10. Comparison of literature in term of \(\text{efficiency}\).
Fig. 1. The proposed System model

Fig. 2. A typical workflow [11].
Fig. 3.a) D-GWO scheduler, makespan=118

Fig. 3.b) PEFT scheduler [11], makespan=122
Fig. 3.c) MPQGA scheduler [5], makespan=122

Fig. 3.d) S-CA scheduler [23], makespan=124
Fig. 3.e) HEFT-upward scheduler [10], makespan=133

Fig. 3.f) D-PSO scheduler [42], makespan=122

Fig. 3. An Illustrative Example

A Wolf Representation:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Fig. 4. An encoded wolf
Algorithm 1. D-GWO-Scheduler

Input:
- A given DAG application with its specification
  - VMs: \{VM_1, VM_2, ..., 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 \(Token_i=(b_{i1}, b_{i2}, ..., b_{in})=\overline{1}\) and \(Adjuster_i=(a_{i1}, a_{i2}, ..., a_{in})=\overline{1}\). (* 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 \(\leftarrow W_\alpha\); BestSolution \(\leftarrow\) 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 \(\leftarrow W_i\)

14: BestSolution \(\leftarrow\) Fitness\((W_i)\)

15: end-if

---(* Exploitation *)---------------------------------------------

16: \(Token_i = Token_i \otimes Adjuster_i\)

---(* Exploration *)---------------------------------------------

17: Draw integer \(q \sim [1..3]\) for Walking Around procedures

18: if \(R=1\) then

19: Call Algorithm 5 for Permutation\(_i(W_i)\)
```plaintext
20:   elseif if R=2 then
21:       Call Algorithm 6 for Permutation2(W_i)
22:   else
23:       Call Algorithm 7 for Permutation3(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_α, W_β, and W_δ three best wolves based on updated fitness values
30:   end for
31:   end while
32:   return BestSoFar and BestSolution
33: End {Algorithm 1}
```

**Fig. 5. Proposed D-GWO**

**Algorithm 2. Initialize Wolves**

**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, ..., W_m) where each i-th wolf is W_i=(W_{i1}, W_{i2}, ..., 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, ..., 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}

**Fig. 6. Generating initial wolves**

**Algorithm 3. Fitness Function**

**Input:**
- W_i: A wolf;
- n: number of tasks in W;
- VMs: \{VM_1, VM_2, ..., 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}  

Fig. 7. Fitness calculation

Algorithm 4. Encircling the Prey  

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 : $\{VM_1, VM_2, ..., VM_M\}$  

Output:  
A new updated wolf $W_i$

1: Let $F_1=\text{Fitness}(W_\alpha)$, $F_2=\text{Fitness}(W_\beta)$, and $F_3=\text{Fitness}(W_\delta)$  
(* $F_1 \leq F_2 \leq F_3$ *)  
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 $P_3=\frac{F_3}{F_1+F_2+F_3}$  
(* $P_1 \leq P_2 \leq P_3$ 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: $Adjuster_i(j) \leftarrow TokenDiff_1(j)$  
else if $q < P_2$ Then  
10: $Adjuster_i(j) \leftarrow TokenDiff_2(j)$  
else  
11: $Adjuster_i(j) \leftarrow TokenDiff_3(j)$  
12: end-if  
13: End-For  
14: $Adjuster_i(1)=Adjuster_i(n)=0$;  
15: Update $W_i$ for encircling the prey by updating its trajectory following $Adjuster_i$  
16: (* Exchange $t_j$ and $t_k$ where $Adjuster_i(t_j)=Adjuster_i(t_k)=1$ and $t_j \neq t_k$ *)  
17: return $W_i$  
18: End {Algorithm 4}
Algorithm 5. *Permutation*₁ procedure

**Input:**

\[ W_i[1..n] : \text{A wolf} \]

\[ n : \text{number of tasks in a wolf} \]

**Output:**

\[ W_i[1..n] : \text{A modified wolf} \]

1: Draw a random integer R in ~ [2..n-1]
2: \[ Z \leftarrow W_i[R] \]
3: \[ t_y \leftarrow \text{Find task } t_y \text{ in set } \{ \text{Succ}(t_Z) \} \text{ which appears ahead in the } W_i[1..n] \text{ where } y=W_i[S]\]
4: Find random \( j \in [R+1..S-1] \) and \( t_p=W_i[j] \); so that the last item of \( \text{Pred}(t_q) \) 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 6. *Permutation*₂ procedure

**Input:**

\[ W_i[1..n] : \text{A wolf} \]

\[ n : \text{number of tasks in a wolf} \]

**Output:**

\[ W_i[1..n] : \text{A modified wolf} \]

1: Draw a random integer R in ~ [2..n-1]
2: \[ Z \leftarrow W_i[R] \]
3: \[ t_y \leftarrow \text{Find task } t_y \text{ in set } \{ \text{Pred}(t_Z) \} \text{ which appears latter in the } W_i[1..n] \text{ where } y=W_i[S]\]
4: Find random index \( j \in [S+1..R-1] \) and \( t_p=W_i[j] \); so that the first item of \( \text{Succ}(t_q) \) 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}

Fig. 11. Walking Around Permutation$_2$

![Diagram showing the exchange process between two tasks in consecutive levels and the resulting permutation]

Fig. 12. Performance of Permutation$_2$

Algorithm 7. Permutation$_3$ procedure

Input:  
- $G$ : 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($t_x$) $\neq$ Level($t_y$)
3: Exchange($W_i[j], W_i[k]$) /* $t_x \leftrightarrow t_y$
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}

Fig. 13. Walking Around Permutation$_3$
Fig. 14. Performance of Permutation$_{3}$

Fig. 15. Molecular workflow [10, 17, 36].
Fig. 16. Performance comparison D-GWO against others in term of *makespan*
Fig. 17. Performance comparison D-GWO versus others in term of SLR

Fig. 18. Performance comparison D-GWO versus others in term of speedup
![Fig. 19. Performance comparison D-GWO versus others in term of efficiency.](image)

**Table 1.** The summary of literature

<table>
<thead>
<tr>
<th>Author(s)/ Ref.</th>
<th>Classification</th>
<th>Approach</th>
<th>Advantages</th>
<th>Shortcomings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topcuoglu et al.[10]</td>
<td>List-based Scheduling Algorithm</td>
<td>HEFT</td>
<td>It quickly ranks tasks and provisions a fast ordered list that preserves precedent constraints.</td>
<td>In large scale problems, it ignores to explore other promising possible solutions in search space.</td>
</tr>
<tr>
<td>Arabnejad and Barbosa [11]</td>
<td>List-based Scheduling Algorithm</td>
<td>PEFT</td>
<td>Similar to HEFT, it provides a topological sorting list of tasks based on cost table prediction.</td>
<td>It does not take VM availability in the course of scheduling. In addition, it neglects other possible solutions.</td>
</tr>
<tr>
<td>Guo and Xuo [31]</td>
<td>List-based Scheduling Algorithm</td>
<td>CEFT</td>
<td>It efficiently compromises between cost and deadline in the course of scheduling.</td>
<td>It is a costly procedure which is solely applicable in very faulty systems because it sometimes reschedule to reach reliability.</td>
</tr>
<tr>
<td>Darbha and Agrawal [32]</td>
<td>Heuristic-based Algorithm</td>
<td>Duplication</td>
<td>It increases parallelism by</td>
<td>It may be costly because it utilizes</td>
</tr>
</tbody>
</table>
duplicate the execution of critical tasks on different VMs. more resources and may charge users more money for residual VMs usage.

It is beneficial method for communication-intensive workflows to reduce data transmission costs. It may possibly reduce the total execution time.

It efficiently explores search space globally. If the degree of heterogeneity is high, it degrades the system performance.

It is very fast approach. It suffers from early convergence and usually gets stuck in local optimum.

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

<table>
<thead>
<tr>
<th>Task</th>
<th>VM1</th>
<th>VM2</th>
<th>VM3</th>
<th>$E^T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>22</td>
<td>21</td>
<td>36</td>
<td>26.3</td>
</tr>
<tr>
<td>T2</td>
<td>22</td>
<td>18</td>
<td>18</td>
<td>19.3</td>
</tr>
<tr>
<td>T3</td>
<td>32</td>
<td>27</td>
<td>43</td>
<td>34.0</td>
</tr>
<tr>
<td>T4</td>
<td>7</td>
<td>10</td>
<td>4</td>
<td>7.0</td>
</tr>
<tr>
<td>T5</td>
<td>29</td>
<td>27</td>
<td>35</td>
<td>30.3</td>
</tr>
<tr>
<td>T6</td>
<td>26</td>
<td>17</td>
<td>24</td>
<td>22.3</td>
</tr>
<tr>
<td>T7</td>
<td>14</td>
<td>25</td>
<td>30</td>
<td>23.0</td>
</tr>
<tr>
<td>T8</td>
<td>29</td>
<td>23</td>
<td>36</td>
<td>29.3</td>
</tr>
<tr>
<td>T9</td>
<td>15</td>
<td>21</td>
<td>8</td>
<td>14.7</td>
</tr>
<tr>
<td>T10</td>
<td>13</td>
<td>16</td>
<td>33</td>
<td>20.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PEFT</td>
</tr>
<tr>
<td>T1</td>
<td>72.7</td>
</tr>
<tr>
<td>T2</td>
<td>41.0</td>
</tr>
<tr>
<td>T3</td>
<td>37.0</td>
</tr>
<tr>
<td>T4</td>
<td>43.7</td>
</tr>
<tr>
<td>T5</td>
<td>31.0</td>
</tr>
<tr>
<td>T6</td>
<td>41.7</td>
</tr>
<tr>
<td>T7</td>
<td>17.0</td>
</tr>
</tbody>
</table>
Table 4. Ordered list of tasks produced by comparative algorithms along with final *makespan*

<table>
<thead>
<tr>
<th>No.</th>
<th>Approach/ Ref.</th>
<th>Generated list of tasks</th>
<th>Final makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HEFT-upward [10]</td>
<td>{ t_1, t_5, t_6, t_2, t_3, t_10 }</td>
<td>133</td>
</tr>
<tr>
<td>2</td>
<td>HEFT-downward [10]</td>
<td>{ t_1, t_5, t_6, t_2, t_3, t_10 }</td>
<td>136</td>
</tr>
<tr>
<td>3</td>
<td>HEFT-level [10]</td>
<td>{ t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10} }</td>
<td>143</td>
</tr>
<tr>
<td>4</td>
<td>PEFT [11]</td>
<td>{ t_1, t_4, t_6, t_2, t_3, t_5, t_8, t_7, t_9, t_{10} }</td>
<td>122</td>
</tr>
<tr>
<td>5</td>
<td>MPQGA [5]</td>
<td>{ t_1, t_5, t_6, t_4, t_2, t_3, t_7, t_8, t_9, t_{10} }</td>
<td>122</td>
</tr>
<tr>
<td>6</td>
<td>Customized–SA (CSA) [23]</td>
<td>{ t_1, t_3, t_5, t_4, t_2, t_6, t_7, t_8, t_9, t_{10} }</td>
<td>124</td>
</tr>
<tr>
<td>7</td>
<td>D-PSO [42]</td>
<td>{ t_1, t_6, t_5, t_4, t_2, t_3, t_7, t_8, t_9, t_{10} }</td>
<td>122</td>
</tr>
<tr>
<td>8</td>
<td>Proposed D-GWO</td>
<td>{ t_1, t_4, t_3, t_6, t_7, t_2, t_5, t_9, t_8, t_{10} }</td>
<td>118</td>
</tr>
</tbody>
</table>

Table 5. Different simulation datasets.

<table>
<thead>
<tr>
<th>CCR</th>
<th>Communication Cost</th>
<th>Computation Cost</th>
<th>Graph Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>[2..10]</td>
<td>[2..15]</td>
<td>computation-intensive</td>
</tr>
<tr>
<td>1.0</td>
<td>[2..10]</td>
<td>[2..10]</td>
<td>Moderate</td>
</tr>
<tr>
<td>5.0</td>
<td>[5..20]</td>
<td>[2..5]</td>
<td>rather communication-intensive</td>
</tr>
<tr>
<td>10.0</td>
<td>[10..40]</td>
<td>[2..5]</td>
<td>communication-intensive</td>
</tr>
</tbody>
</table>

Table 6. Parameter Settings of Comparative Algorithms.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Specific parameters</th>
<th>Population Size</th>
<th>Max Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPQGA [5]</td>
<td>Crossover Percentage: 0.8</td>
<td>50–150 depends on scenario</td>
<td>100–150 depends on scenario</td>
</tr>
<tr>
<td></td>
<td>Mutation Percentage: 0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-SA [23]</td>
<td>Freeze ( T_0 )</td>
<td>0</td>
<td>Point-wise</td>
</tr>
<tr>
<td></td>
<td>( \Delta T )</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>D-PSO [42]</td>
<td>( C_1 = C_2 )</td>
<td>1.5</td>
<td>50–100 depends on scenario</td>
</tr>
<tr>
<td></td>
<td>( \omega )</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V_{\text{Max}} )</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

\( T_0 \): Initial Temperature  
\( \Delta T \): Decrease Temperature Amount  
**Freeze:** Freeze Temperature for Final Condition
\[ V_{\text{Max}} \] Velocity limit for clamping
\[ C_1 \] Personal acceleration coefficient
\[ C_2 \] Social acceleration coefficient
\[ \omega \] Inertia coefficient

Table 7. Comparison of literatures in term of \textit{makespan}.

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of VMs</th>
<th>\textit{makespan}</th>
<th>RPD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PEFT</td>
<td>C-SA</td>
</tr>
<tr>
<td>0.4</td>
<td>3</td>
<td>215</td>
<td>210</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>178</td>
<td>175</td>
</tr>
<tr>
<td>1.0</td>
<td>3</td>
<td>106</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>5.0</td>
<td>3</td>
<td>136</td>
<td>131</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>123</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>10.0</td>
<td>3</td>
<td>137</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>137</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>137</td>
<td>135</td>
</tr>
</tbody>
</table>

Table 8. Comparison of literature in term of \textit{SLR} metric.

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of VMs</th>
<th>\textit{SLR}</th>
<th>RPD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PEFT</td>
<td>C-SA</td>
</tr>
<tr>
<td>0.4</td>
<td>3</td>
<td>1.99</td>
<td>1.94</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.69</td>
<td>1.69</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1.65</td>
<td>1.62</td>
</tr>
<tr>
<td>1.0</td>
<td>3</td>
<td>2.79</td>
<td>2.71</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.45</td>
<td>2.45</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.37</td>
<td>2.37</td>
</tr>
<tr>
<td>5.0</td>
<td>3</td>
<td>5.23</td>
<td>5.04</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4.73</td>
<td>4.92</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4.73</td>
<td>4.73</td>
</tr>
<tr>
<td>10.0</td>
<td>3</td>
<td>6.23</td>
<td>6.18</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6.23</td>
<td>6.14</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6.23</td>
<td>6.14</td>
</tr>
</tbody>
</table>

Table 9. Comparison of literature in term of \textit{speedup}.

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of VMs</th>
<th>\textit{speedup}</th>
<th>RPD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PEFT</td>
<td>C-SA</td>
</tr>
<tr>
<td>0.4</td>
<td>3</td>
<td>2.34</td>
<td>2.40</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.75</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.83</td>
<td>2.88</td>
</tr>
<tr>
<td>1.0</td>
<td>3</td>
<td>2.08</td>
<td>2.14</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.37</td>
<td>2.37</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.44</td>
<td>2.44</td>
</tr>
<tr>
<td>5.0</td>
<td>3</td>
<td>1.00</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.11</td>
<td>1.06</td>
</tr>
</tbody>
</table>
Table 10. Comparison of literature in term of efficiency.

<table>
<thead>
<tr>
<th>CCR</th>
<th>No. of VMs</th>
<th>efficiency (%)</th>
<th>RPD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PEFT</td>
<td>C-SA</td>
</tr>
<tr>
<td>0.4</td>
<td>3</td>
<td>78.00</td>
<td>80.00</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>55.00</td>
<td>55.00</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>40.43</td>
<td>41.14</td>
</tr>
<tr>
<td>1.0</td>
<td>3</td>
<td>69.33</td>
<td>71.33</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>47.40</td>
<td>47.40</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>34.86</td>
<td>34.86</td>
</tr>
<tr>
<td>5.0</td>
<td>3</td>
<td>33.33</td>
<td>34.67</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>22.20</td>
<td>21.20</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>15.86</td>
<td>15.86</td>
</tr>
<tr>
<td>10.0</td>
<td>3</td>
<td>33.33</td>
<td>33.67</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>20.00</td>
<td>20.20</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>14.29</td>
<td>14.43</td>
</tr>
</tbody>
</table>