Using Differential Evolution and Moth-Flame Optimization for Scientific Workflow Scheduling in Fog Computing

Omed Hassan Ahmed[1,2], Joan Lu[1], Qiang Xu[1], Aram Mahmood Ahmed[3,4], Amir Masoud Rahmani[5*], and Mehdi Hosseinzadeh[6*]
[1]School of Computing and Engineering, University of Huddersfield, Huddersfield, England
[2]University of Human Development,College of Science and Technology, Department of Information Technology,Sulaymaniyah,Iraq
[3]Department of Information Technology, Sulaimani Polytechnic University, Sulaymaniyah, Iraq
[4]International Academic Office, Kurdistan Institution for Strategic Studies and Scientific Research, Sulaymaniyah, Iraq
[5]Future Technology Research Center, National Yunlin University of Science and Technology, Yunlin, Taiwan
[6]Pattern Recognition and Machine Learning Lab, Gachon University, 1342 Seongnamdaero, Sujeonggu, Seongnam 13120, Republic of Korea
omed.ahmed@hud.ac.uk[1,2], j.lu@hud.ac.uk[1], q.xu2@hud.ac.uk[1], aram.ahmad@spu.edu.iq[3,4], rahmania@yuntech.edu.tw[5], mehdi@gachon.ac.kr[6]
Corresponding authors (Amir Masoud Rahmani, and Mehdi Hosseinzadeh)

Abstract: Fog computing is an interesting technology aimed at providing various processing and storage resources at the IoT networks' edge. Energy consumption is one of the essential factors that can directly impact the maintenance cost and CO_2 emissions of fog environments. Energy consumption can be mitigated by effective scheduling approaches, in which tasks are going to be mapped on the best possible resources regarding some conflicting objectives. To deal with these issues, we introduce an opposition-based hybrid discrete optimization algorithm, called DMFO-DE. For this purpose, first, a discrete and Opposition-Based Learning (OBL) version of the Moth-Flame Optimization (MFO) algorithm is provided, and it then is combined with the Differential Evolution (DE) algorithm to improve the convergence speed and prevent local optima problem. The DMFO-DE is then employed for scientific workflow scheduling in fog computing environments using the Dynamic Voltage and Frequency Scaling (DVFS) method. The Heterogeneous Earliest Finish Time (HEFT) algorithm is used to find the tasks execution order in the scientific workflows. Our workflow scheduling approach mainly tries to decrease the scheduling process's energy consumption by minimizing the applied Virtual Machines (VMs), makespan, and communication between dependent tasks. For evaluating the performance of the proposed scheduling scheme, extensive simulations are conducted on the scientific workflows with four different sizes. The experimental results indicate that scheduling using the DMFO-DE algorithm can outperform other metrics such as the number of applied VMs, and energy consumption.

Keywords: Fog Computing, Task, Workflow, Optimization, Makespan, Energy, DVFS.

1. Introduction:

Internet of things (IoT) is a collection of devices or things such as various home appliances [1, 2] that contain the required software and hardware technologies for communicating with other things and systems [3]. These things can gather monitoring data from their environment and are able to cooperate using unique addressing methods [4]. As a result, IoT is successfully integrated with other contexts such as smart homes, smart cities, e-healthcare, transportation, industries, agriculture, etc. However, IoT things often suffer from low computational, storage, and power resources [5, 6]. Cloud computing [7] is a well-known technology that can deliver the required resources by the IoT from. Typically, cloud computing benefit from a large number of VMs in its data centers and using proper resource management techniques such as VM migration[8] and VM placement[9-11], can deliver the required resources by the IoT. However, using cloud computing resources incurs long delays, which may be unacceptable in some applications. Therefore, there is a desperate need for more resources at the edge of the IoT networks. Fog computing is new technology to answer such requirements and is able to provide the required resources to facilitate task processing, networking, and data storage[12] near the IoT networks [13]. Thus, fog computing can effectively reduce the communication overheads and delay for the IoT networks while mitigating the workloads on cloud computing data centers [14, 15]. Each fog server is a virtualized environment equipped with computational devices, a wireless communication unit, and data storage cards. Many VMs can serve requests from mobile users, and IoT devices resided in the fog servers. A mobile user can communicate directly with fog servers through a single-hop wireless connection using a wireless interface, namely 4G LTE devices, WiFi, Bluetooth, etc. When fog computing receives more requests than it can handle, it may offload some of its load to the cloud computing data centers, as needed [10, 16, 17]. Figure 1 depicts fog computing architecture in which, in the bottom layer, IoT devices are located [18].

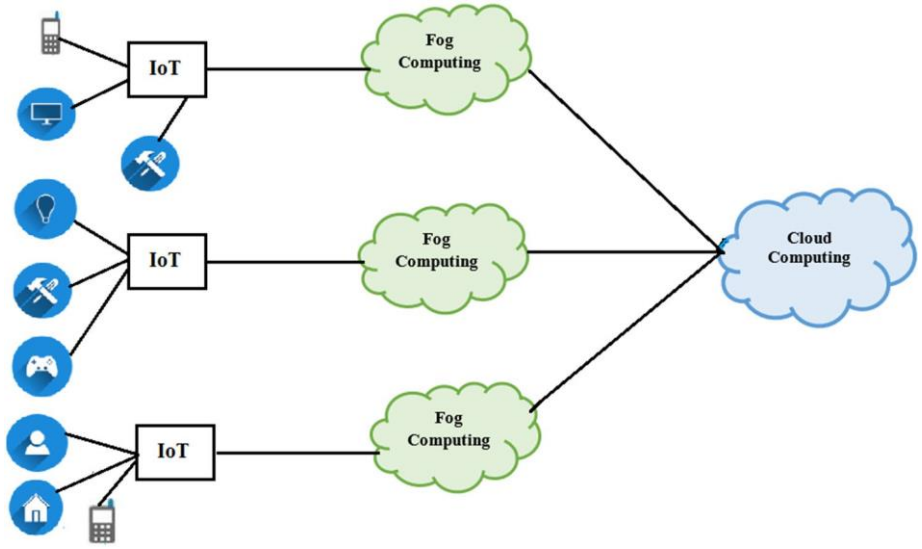


Figure 1: Fog computing architecture

Then, fog computing resources are placed at the top of the IoT devices, and the top layer is Cloud computing, which is a paradigm for hosting services over the Internet. Efficient resource management[19] is one of the essential requirements of fog computing and can help provide high-quality IoT services. Task and workflow scheduling are practical resource management approaches in the fog computing environments to assign a set of tasks requested by the IoT to the most appropriate fog nodes [20-22]. Such fog scheduling schemes try to manage various fog virtual resources and focus on user-specified constraints and deadlines. Also, scheduling approaches often minimize the makespan, resource consumption, data transfer, and cost while ensuring the quality of service factors such as deadlines[23, 24]. Task and workflow scheduling problems are proven to be NP-hard problems in fog computing [25-27]. In this context, several heuristics and metaheuristic scheduling methods are widely studied by scheduling independent tasks. Still, fewer researches have been presented for workflow scheduling in the fog computing context. Based on their applied algorithm, workflow scheduling schemes can be classified as heuristic[28-38] and metaheuristic schemes, which in the latter case different optimization algorithms, such as particle swarm optimization, genetic algorithm, etc., are used in the metaheuristic fog scheduling approaches. For instance, in [39], Hosseinioun et al. introduced a DVFS-based power-aware approach to save energy in fog computing, which schedules tasks in lower voltage and frequency in slack times. They used IWO-CA, a hybrid evolutionary algorithm to order tasks regarding task precedence. Also, in the scheduling approach provided in [40], the authors presented the TCaS algorithm based on an evolutionary algorithm. They indicated that it could reduce the makespan and execution cost while meeting users' QoS and cost requirements. Besides, in [41], Aburukba et al. schedule the IoT tasks to reduce their execution latency in fog-cloud computing. This scheme considers delays like waiting time, routing or queuing delay, transmission delay, and processing time for processing IoT requests. Furthermore, they used genetic algorithms to find the right solutions quickly and indicated that it improves the performance and latency while meeting the deadlines. Also, the scheduling scheme provided in [42] applies the MFO algorithm to assign an optimal set of fog nodes to tasks to meet the applications' QoS requirements and mitigate their makespan. The authors have conducted their experiments using the iFogSim toolkit and indicated that their scheme could reduce the execution time. Moreover, in [43, 44], the bees life algorithm, which is an optimization approach for workflow scheduling in the fog is presented, which optimizes the task distribution among fog nodes. This scheme is aimed to optimize the CPU execution time and allocated memory. Also, in [45], Maio and Kimovski proposed an NSGA-II-based scheduling approach for data-intensive workflows, which considers the response time, cost, and reliability. They extend a Fog simulation framework to evaluate the cloud offloading suitability for data-intensive scientific workflows. They indicated that the use of fog is beneficial for low computation and high data transmission tasks, making it the best choice for optimizing the response time of data-intensive tasks. They indicated that their approach could reduce response time compared to HEFT for small task sizes while slightly maintaining cost and reliability values. Besides, in [46], a particle swarm optimization-based cost-aware scheduling approach for multiple workflows is presented.

These scheduling approaches consider various factors such as makespan, communication delay, cost, reliability[47], security[48-50], etc. Furthermore, energy consumption is one of the critical factors considered in scheduling schemes[51]. Different techniques are used in the literature to reduce the fog computing environments' energy

consumption, in which DVFS is one of them. DVFS is an interesting method that can reduce the processor's frequency to mitigate its power usage. However, reducing the processor's frequency increases its execution time, and as a result, the primary deadline and various Quality of Service (QoS) factors should not be neglected. Although numerous researches have been carried out in the DVFS-based scheduling context in cloud computing platforms, very few DVFS-based schemes are provided in the literature for fog computing environments. Thus, conducting further investigation in the energy-efficient scheduling in the fog computing environments is an essential issue.
In the optimization algorithm context, the moth-flame optimization algorithm or MFO is an interesting algorithm introduced by Mirjalili [52]. This algorithm is inspired by the movement of the moths for solving continuous optimization problems. However, despite the MFO's success in the continuous optimization context, it cannot handle discrete problems without modification. For this purpose, in this article, we present DMFO, a discrete version of the MFO algorithm, which uses genetic operators such as mutation and crossover to produce discrete solutions. But, the MFO algorithm suffers from the low convergence speed and global search capability [53]. To handle these issues, opposition-based learning (OBL) is used in the DMFO algorithm. The DMFO is combined with the DE algorithm to provide a new hybrid, and discrete optimization algorithm denoted as DMFO-DE. In each iteration, the DMFO-DE algorithm randomly executes one of the DMFO and DE algorithms. Their achieved results give a penalty or reward to these algorithms using a Learning Automaton (LA). Using this technique, the algorithm that gets better results achieves more chance to explore the problem space. However, to give both algorithms a chance to find better solutions, every 30 rounds, the LA used for each algorithm is reset, increasing the exploration of the proposed algorithm. The DMFO-DE algorithm is also used for DVFS-based scheduling of scientific workflows submitted to mitigate the fog computing environment's energy consumption in the scheduling process. Furthermore, in this scheduling approach, different chaotic maps are used to produce a discrete initial population. Besides, in this scheme, the HEFT scheduling algorithm's task prioritization method is applied for finding the order of task execution in the scientific workflows. Finally, to verify the proposed scheduling schemes' energy efficiency, several experiments with different size scientific workflows are conducted. Our primary contributions in this paper can be listed as follows:

- Presenting a discrete version of the MFO algorithm, denoted as DMFO.
- Presenting DMFO-DE, a hybrid and OBL-based optimization algorithm using the DMFO and DE optimization algorithms.
- Introducing a scientific workflow scheduling scheme using the DMFO-DE algorithm.
- Effective energy management using DVFS in the introduced workflow scheduling approach.
- Extensive simulations using the iFogSim simulator tool to verify the proposed algorithm's effectiveness in terms of energy consumption and the number of applied VMs, with different DVFS settings.

The rest of this scheduling article is organized as follows: Section 2 gives an interesting review of the fog computing domain's previously published scheduling approaches. Section 3 discusses the MFO algorithm. Section 4 presents the proposed discrete version of the MFO algorithm, and Section 5 introduces our proposed workflow scheduling framework. Moreover, Section 6 reports the experimental results, and finally, Section 7 puts forward the concluding remarks and future study directions.

2. MFO algorithm

MFO algorithm is a population-based algorithm inspired by the moths' movement and is used for solving continuous optimization problems. Typically, moths fly by keeping a fixed angle to the moon that is an efficient method for traveling in a straight path, but artificial lights trick the moths and show such behaviors. Because the light is close to the moth, keeping a similar angle to the light source causes a moths' spiral fly path. In the MFO, the set of moths is represented in a matrix as follows:

$$Moth = \begin{matrix} Moth_{11} & \cdots & Moth_{1d} \\ \vdots & \ddots & \vdots \\ Moth_{n1} & \cdots & Moth_{nd} \end{matrix} \quad (1)$$

In which *n* determines the number of moths and *d* denotes their dimension. Besides, the fitness of the moths should be maintained as follows:

$$MothF = \begin{matrix} MothF_1 \\ MothF_2 \\ \dots \\ MothF_n \end{matrix} \quad (2)$$

Flames are the other components of the proposed MFO algorithm and are stored in a matrix as follows:

$$Flame = \begin{matrix} Flame1,1 & \cdots & Flame1,d \\ \vdots & \ddots & \vdots \\ Flamen,1 & \cdots & Flamen,d \end{matrix} \quad (3)$$

Which *n* specifies the number of flames, and *d* determines the number of dimensions. Furthermore, it is assumed that there is an array for storing the corresponding fitness values as follows:

$$FlameF = \begin{matrix} FlameF_1 \\ FlameF_2 \\ \dots \\ FlameF_n \end{matrix} \quad (4)$$

In which *n* specifies the number of flames. It should be noted that both moths and flames are solutions, but they differ in their updating method. The moths are search agents that can explore the search space, but flames are the best solutions that have been reached so far. In this algorithm, each moth searches around a flame and updates the moth when it finds a solution with a better fitness value. Figure 2 indicates the population initialization method in the MFO algorithm.

```
for i = 1 to n step 1
    for j = 1 to d step 1
        moth = upperb lowerb * rand + lowerb
    end
end
MothF = FitnessF Moth
```

Figure 2: Population initialization in the MFO algorithm

Furthermore, the upper bound and lower bound of the solutions are defined by two arrays: *upperb* and *lowerb*. These arrays are defined as follows:

$$upperb = \{upperb_1, upperb_2, upperb_3, \dots, upperb_n\} \quad (5)$$

In which, $upperb$ specifies the upper bound of the i[th] dimension of each solution.

$$lowerb = \{lowerb_1, lowerb_2, lowerb_3, \dots, lowerb_n\} \quad (6)$$

In which, $lowerb$ indicates the lower bound of the i[th] dimension of each solution. For mathematically modeling this behavior, the position of each moth is updated concerning a flame using the following equation:

$$Moth = S\ Moth, Flame \quad (7)$$

In this equation, M_i indicates the i[th] moth, and Fj indicates the j[th] flame, and *S* is a logarithmic spiral function used for updating the moths. Based on these issues, a logarithmic spiral is defined for the MFO algorithm as follows:

$$S(Moth, Flame) = Distance \cdot e^{bt} \cos 2\pi t + Fj \quad (8)$$

$Distance$ indicates the distance of the i[th] moth for the j[th] flame, *t* is a random number in [-1, 1], and *b* is a constant for defining the shape of the logarithmic spiral. Furthermore, $Distance$ is calculated as follows:

$$Distance = |Flame \quad Moth| \quad (9)$$

$Moth_i$ indicates the i[th] moth, $Flame_j$ indicates the j[th] flame, and $Distance_i$ is the distance.

3. The proposed discrete optimization algorithm

This section presents our proposed DMFO algorithm and DMFO-DE created by integrating the DE [54] and DMFO algorithms using learning automata.

3.1. OBL

OBL or Opposition-based Learning is a simple method that has been incorporated in several optimization algorithms to enhance their capabilities. In OBL, the opposite of each dimension of the opposition solution can be computed as shown in Equation 10:

$$x^* = a + b \quad x \quad (10)$$

Which a_i and b_i are the upper bound and lower bound of the i[th] dimension of the solution *x*.

3.2. DMFO algorithm

To provide a discrete version of the MFO algorithm, we have to introduce a method to create a discrete initial population and then design the required operators to modify these discrete solutions and convert them to the other discrete solutions[55]. We used an opposition-based method for producing a discrete population, which has been shown in Figure 3. As shown in this figure, we have used the chaotic maps to produce random numbers between [0, 1], and Nvm denotes the maximum number of available VMs in the fog computing environment. Using this method, a random integer number between [0, *Nvm*] is created by each solution's dimension. After creating the required number

of solutions, the opposition of each solution is computed. However, the solution's opposite can be applied when it has a lower fitness value than a chaotic map solution.

```
For i = 1 to n
    For j = 1 to d
        moth i,j = Floor Nvm ∗Chaotic_MAP())
    End
    Temp=OBL(moth i,j
    If Fitness(Temp)< moth_i Then
        moth i,j =Temp
    End
End
```

Figure 3: Population initialization in the DMFO algorithm

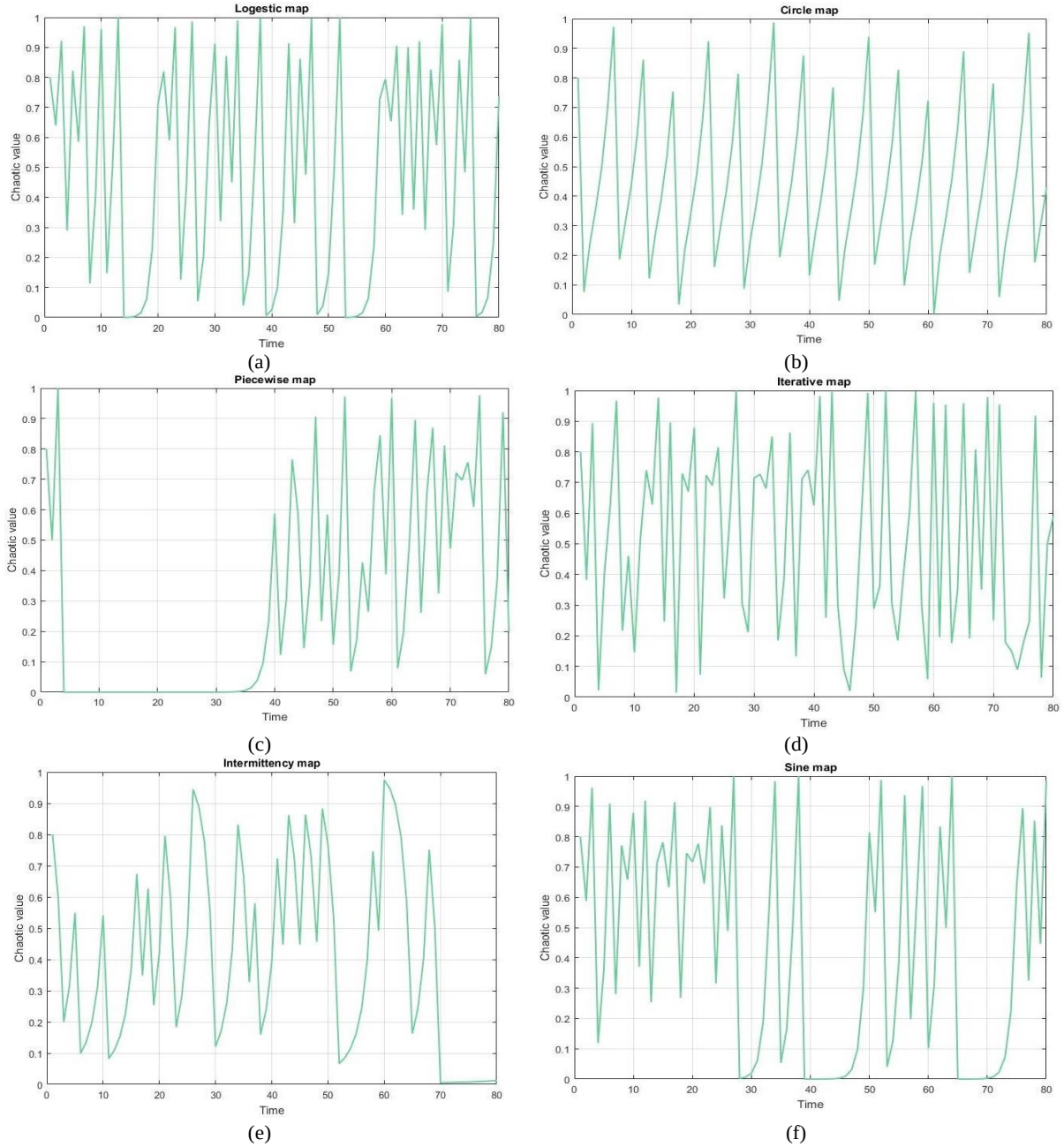


Figure 4: Applied chaotic maps

Figure 4 indicates various chaotic maps [56, 57] applied in the DMFO algorithm, which can be described as follows:

- Figure 4.a shows the logistic map output over 80 iterations, and Equation 11 indicates the logistic map, where a=4.
- Equation 12 shows the circle map, in which P=0.4 is used. Figure 4.b shows the output of this map over 80 iterations.
- Equation 13 defines the piecewise map, in which n=1.6, C=1.5, and P=0.5. Figure 4.c shows chaotic value distributions of this chaotic map for 80 iterations.
- Equation 14 defines the iterative map, where a=0.5 and b=0.2 are used. Figure 4.d shows the chaotic value distributions of this map for 80 iterations.
- Equation 15 indicates the Intermittency map, in which C=1.5, n=1.6, and P=0.5 are used. Figure 4.e depicts the output of the Intermittency map over 100 iterations.
- Equation 16 defines the Sine chaotic map, in which a=4 is used. Figure 4.f shows its values over the 100 iterations.

$$K_{+1} = aX \quad 1 \quad X \tag{11}$$

$$X_{+1} = \begin{cases} \dfrac{X}{P} & 0 \$ X < P \\ \dfrac{X \quad P}{0.5 \quad P} & P \$ X < 0.5 \\ \dfrac{1 \quad P \quad X}{0.5 \quad P} & 0.5 \$ X < 1 \quad P \\ \dfrac{1 \quad X}{P} & 1 \quad P \$ X < 1 \end{cases} \tag{12}$$

$$X_{+1} = \begin{cases} \varepsilon + X + C\ X^{n} & 0 < X \$ P \\ \dfrac{X \quad P}{1 \quad P} & P < X < 1 \end{cases} \tag{13}$$

$$X_{+1} = X + b \quad (\frac{a}{2\pi}) \sin 2\pi X \quad mod\ 1 \tag{14}$$

$$X_{+1} = \sin(\frac{a\pi}{X}) \tag{15}$$

$$X_{+1} = \frac{a}{4} \sin \pi X \tag{16}$$

This section uses genetic operators such as swap, crossover, and mutation to modify the basic MFO algorithm. In this equation, the mutation operator selects a random element for the j^{th} dimension. After M is computed for each $j=1...D$, and M_i is achieved, we should compute Equation 15. For this purpose, as indicated in Equation 36, we use a multi-point crossover operator on the F and M vectors:

$$M = CrossOver(F, M) \tag{17}$$

Equation 18 indicates a two-point crossover operator.

$$M = \begin{matrix} F_{ij} & if\ \theta < Rand \\ M_{ij} & if\ \theta < Rand \\ F_{ij} & if\ \theta\ ?: Rand \end{matrix} \tag{18}$$

Figure 5 indicates the effect of a crossover operator on two sample vectors.

22	10	3	85	62	33	2
54	87	41	20	4	15	82
22	10	3	20	4	15	82

Figure 5: A sample two-point crossover

Figure 6 indicates the DMFO algorithm's pseudo-code, which benefits from the OBL method both in the initial population creation and during the iterations to search for the best solution. In each iteration, for each month, the algorithm may perform the crossover with a flame or with the *Disctance*$_i$. It may also perform the swap operation one each moth or may perform mutation on it.

```
Function DMFO
Input: A random population
Output: The best solution
For i = 1 to n
    For j = 1 to d
        Moth i,j = Floor Nvm ∗Chaotic_MAP())
    End
    Temp=OBL(moth i,j
    If Fitness(Temp)< Moth_i Then
        Moth i,j =Temp
    End
End
For k=1: Max_Iteration
    For i = 1 to n
        Calculate Distance(i)
        Z= Chaotic_MAP()
        If (Z<=0.25)
            // Swapping the Moth and Flame
            Xc1=Rand(1, n)
            Moth(i)=Crossover(flame(i),Moth(i), Xc);
        End
        If (0.25<=Z<0.5)
            // Swapping the Moth and Flame
            Xc2=Rand(1, n)
            Moth(i)=Crossover(flame(i), Distance(i), Xn);
        End
        If (0.5<=Z<0.75)
            // Computing M(i)= swap(flame i );
            Ns=Rand(1,n)
            For i=1 to Nswap
                Xj1= Rand(1,n)
                Xj2= Rand(1,n)
                Swap(Moth(i, Xj1),Moth(i, Xj2)
            End
        End
        If (Z>=0.75)
            // Computing M(i)=Mutate(flame(i));
            For i=1 to Nmutate
                Xj= Rand(1,n)
                Moth(i,Xj)= Floor Nvm ∗Chaotic_MAP())
            End
        End
        OBL(i)= Opposition of the Moth(i)
        If Fitness(OBL(i))< Moth i Then
            Moth i = OBL(i)
        End
    End
End
```

Figure 6: DMFO pseudo-code

3.3. DE algorithm

Differential evolution or DE is an easy to understand metaheuristic algorithm that iteratively searches the optimization problems' solution space to find the best solution. It works by having a population of candidate solutions that move around in the search space using a simple mathematical formula to combine existing agents' positions from the population. If the new position is an improvement, then it is accepted and forms part of the population. Otherwise, the new position is simply discarded. The process is repeated, and by doing so is hoped, but not guaranteed, that a

satisfactory solution will eventually be discovered. In the mutation phase, the solutions are updated using the following equation:

$$V\ t+1\ = X_1\ t\ + F.\ X_2\ t\ \quad X\ \ t \tag{19}$$

Where $r_1, r_2, r \in \{1,2, \dots, N\}$ are random numbers, and F is a constant between $[0, 2]$ and controls the amplification of the differential variation. In the crossover phase, the position of the solutions is updated as below:

$$U\ \ t+1\ = \begin{cases} V\ \ t+1 & if\ \ randb\ j\ \$\ CR\ \ or\ j = rnbr\ \ i \\ X\ \ t & if\ \ randb\ j\ \ > CR\ \ or\ j = \mathrm{t}\ rnbr\ \ i \end{cases},\ \ j = 1,2, \dots, D \tag{20}$$

Where j indicates the dimension, $randb$ is a vector with random values in (0,1), CR is crossover constant, and $rnbr$ is a randomly chosen index. At last, to decide which solutions should become a member of generation $t+\ 1$, the trial vector $U\ \ t\ +\ 1$ is compared to the base solution $X\ \ t$ using the greedy selection mechanism. If the solution $U\ \ t\ +\ 1$ has a better cost than $X\ t$, then the U (t+1) replaces with $X\ t+1$ otherwise, the old $X\ t$ is kept.

3.4. Learning Automata

Typically, a learning automaton (LA) can be considered as a machine that can stochastically perform a finite number of actions. This environment responds to it the learning automata after evaluating the action, and these responses should be used in the LA as feedback to select its next action[58]. The LA uses this response, updates its internal state, and learns how to choose the best next action. With this process, the automaton gradually discovers the best action within its set. Generally, LA can be defined as (S, A, P, δ, ω), where $S = \{s_1, s_2, s_3, \dots, s_n\}$ specifies the set of states of LA, $A = \{a_1, a_2, a_3, \dots, a_n\}$ is the set of actions which LA performs, and $R = \{r_1, r_2, r_3, \dots, r_n\}$ is the set of responses received from the environment. Also, δ is a function that maps the current state of LA and the environment's input to the next state of LA, and ω is a function that maps the LA's current state and the environment's response to the LA state. In LA, the environment can have one of the following models:

- *P-Model*: The output can take only two values, 0 or 1.
- *Q-Model*: Finite output set with more than two values, between 0 and 1.
- *S-Model*: The output is a continuous random variable in the range [0, 1].

Also, each LA action may be rewarded or penalized. When the action gets the reward, its probability increases, while all other actions decrease. Generally, the reward and penalty can be computed as follows:

- Reward:

$$P_i(n+1) = P_i(n) + d[1 - P_i(n)] \tag{21}$$

$$P_j(n+1) = (1-d)P_j(n)\ \ For\ all\ j <> i \tag{22}$$

- Penalty:

$$P_i(n+1) = (1-e)P_j(n) \tag{23}$$

$$P_j(n+1) = e/(r-1) + (1-e)P_j(n)\ \ For\ all\ j <> i \tag{24}$$

In the preceding expressions, d is the reward parameter, and e is the penalty parameter.

3.5. Hybrid DMFO-DE algorithm

Typically, the non-hybrid metaheuristic algorithms rely only on some mathematical formula and may fall into local optima problem. For this purpose, hybrid metaheuristic algorithms are proposed in the literature to avoid their constituent algorithms' shortcomings and solve more complex problems. Generally, the following three models are used for integrating the optimization algorithms and creating new hybrid metaheuristic algorithms:

- Serial method: Each of the metaheuristic algorithms will be executed serially, one after another.
- Parallel method: The metaheuristic algorithms will be executed in parallel. In this case, each algorithm may operate on the whole population or a subpopulation. In the latter case, proper consideration should be made for sub-populations creation, integration, and management.
- Conditional method: In this case, based on some conditions, only one of the optimization algorithms will be run in each iteration. This method incurs less overhead than the two previous methods and is used in our proposed hybrid algorithm.

In this scheme, we combine the DE algorithm and our proposed DMFO algorithm to achieve better results. Figure 7 depicts our proposed hybrid optimization protocol's flowchart, which conditionally executes the DE and DMFO algorithms. As shown in this figure, at first, the hybrid algorithm starts with a probability of 0.5 ($P=0.5$). Based on this random value, either DE or the DMFO algorithms should be executed. According to the achieved results, the

learning automata should be used to reward or penalize the executed algorithm. Using this method, the algorithm that has gained some improvements can find more chances to continue improving. However, when each of the algorithms cannot improve after a predefined number of iteration, variable P is used for selecting the algorithm should be reset to its initial value ($P=0.5$) to improve the exploration chance. This operation should be continued until a predefined number of iterations has been reached. Generally, the proposed hybrid DMFO-DE algorithm is lighter than the main MFO because instead of the MFO, the DE algorithm, which is much lighter, should be executed in some rounds.

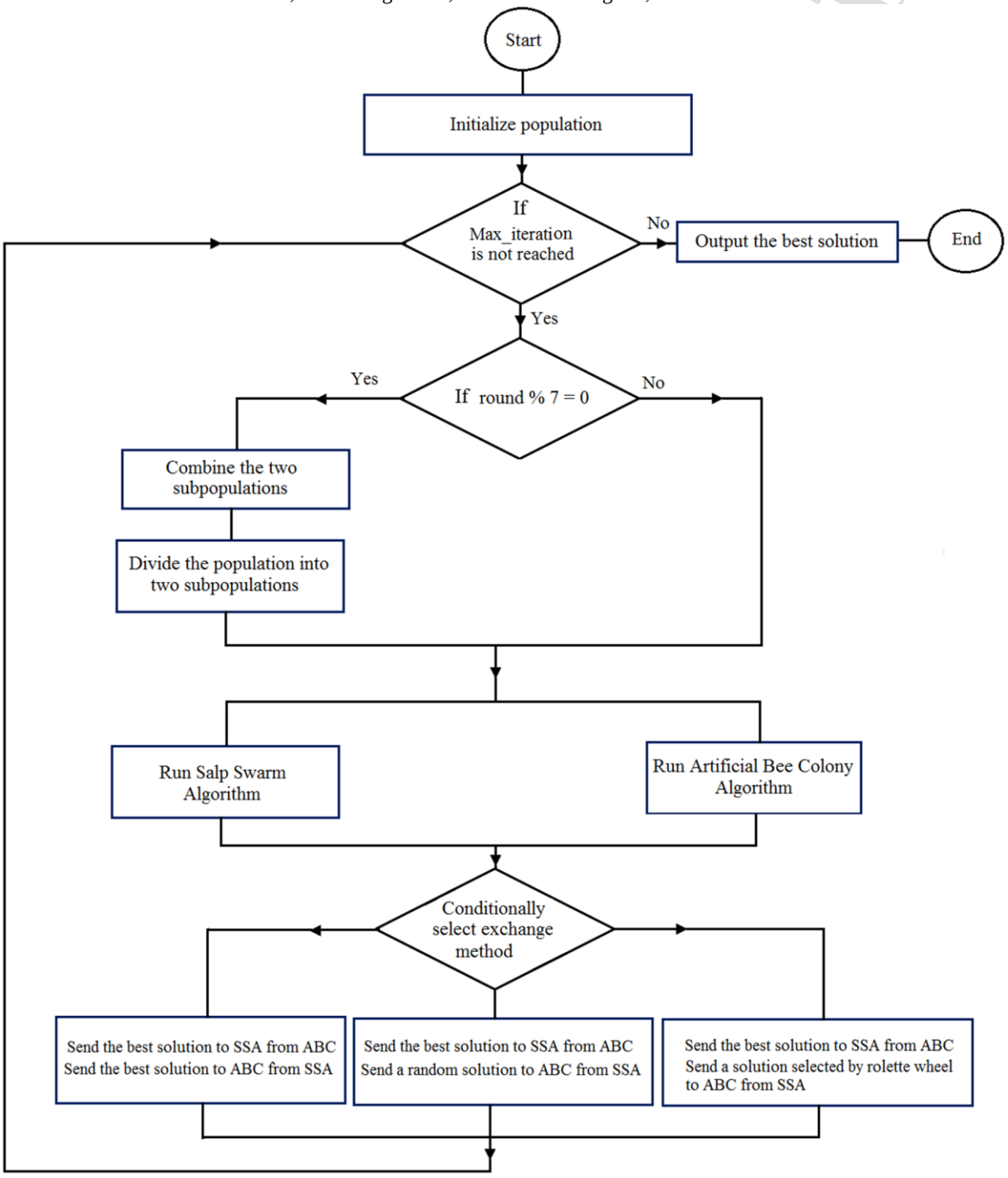


Figure 7: The DMFO-DE algorithm

This reduces the proposed algorithm's execution cost and alleviates the high costs of the sorting algorithms used in the MFO algorithm. Also, the DMFO-DE algorithm does not divide the population into two sub-population (swarm), and in each iteration, each of the applied algorithms works on the whole population. This hybridization method reduces the algorithms overhead than other hybrid optimization techniques, which need each participant algorithm to work a swarm on the result be integrated later. Thus, there will be no need for various swarm management operations necessary for multi-swarm algorithms. Furthermore, it is worth mentioning that in this algorithm, we have also used chaotic maps to produce the required random numbers.

4. Workflow Scheduling Using DMFO-DE Algorithm

This scheme's primary goal is to assign the most appropriate set of VMs to the IoT submitted workflows to minimize both workflow makespan and fog computing energy consumption. This section explains how the proposed DMFO-DE algorithm is used for efficient workflow scheduling in a fog computing environment to achieve such objectives. For this purpose, we first provide a formal formulation of the workflow scheduling problem. Table 1 specifies the abbreviations applied in the rest of this section.

Table 1: Abbreviations and Acronyms

Abbreviation	Description
VM_i	i^{th} VM
Nvm	Number of VMs
W_i	i^{th} workflow
T_i	i^{th} task
$FT\,T\,,VMj$	Finish time of ith task on the jth VM
F_DVFS_i	Dynamic Voltage Frequency Scaling
$Frequency_i$	i^{th} frequency
$Voltage_i$	i^{th} voltage
$avail(VM\,)$	The time that j^{th} VM will be available
$Exection\_Time\,T\,,F\_DVFS_k$	The execution time of the task T_i with the k^{th} level DVFS
$STime\,T\,,VM$	The start time of the task T_i on the VM_j
$FT(T\,,VM\,)$	Finish time of the task T_i on the VM_j
$Ave\ Exection\_Time\ Ti$	The average execution time of the task T_i
$Com\_Time\,T\,,Tj$	Communication time of the data transfer between the T_i and T_j
SlackTime	Slack Time
Predecessor(T_i)	Predecessor set of task T_i
Successor(T_i)	Successor set of task T_i
Wcommunication	The total amount of workflow communications
Scommunication	The required amount of scheduling communications
NT_i	number of tasks in the i^{th} level of the workflow
E_{busy}	The energy consumption of the VMs
E_{Idle}	The energy consumption of idle periods

4.1. DVFS

Typically, the DVFS method reduces the CPU's operational frequency and voltage to mitigate processors' energy consumption in the task execution. DVFS method can be incorporated in all computing systems, ranging from mobile devices to cloud computing DCs. However, reducing the CPU frequency mitigates its speed, and as a result, the workflow deadline may be missed unexpectedly. Thus, the DVFS-based scheduling frameworks' main task is to determine the minimum necessary operating frequency for the VMs that execute the workflow while meeting the workflow deadline.

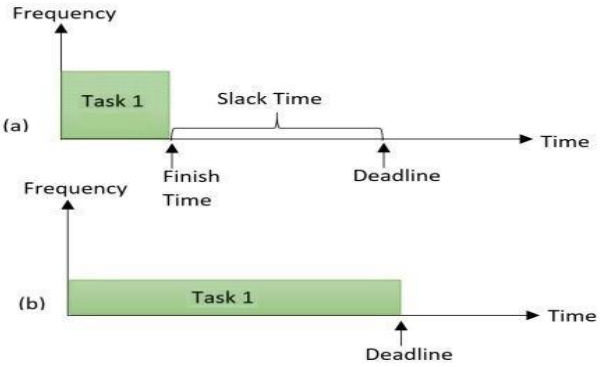


Figure 8: Slack time

As depicted in figure 8, when more frequency is used in DVFS-based scheduling [59, 60], the task is executed faster, but when a lower level of frequency is utilized, it takes longer to run the task. Figure 8.a shows the slack time, which can be used by DVFS-based scheduling. As shown in Equation 25, the slack time is the period between the deadline ($TaskDeadline$) and the task finish time ($TaskFinishTime$ ($F_{\max}$)), when the processor is run with maximum possible frequency.

$$SlackTime = TaskDeadline - TaskFinishTime\ (F_{\max}) \quad (25)$$

Figure 9 depicts some parts of a workflow that perform data aggregation and should be scheduled on the fog computing environment. As shown in this figure, T1, T2, T3, T4, and T5 should be executed and completed before task T6 starts to be executed.

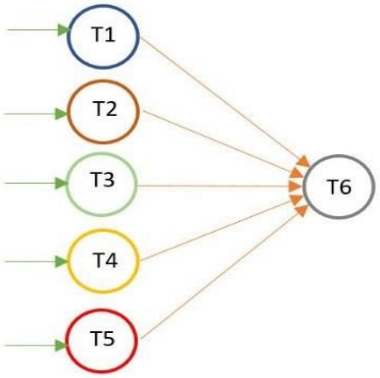


Figure 9: Data aggregation point in a workflow

However, in some cases, T1 to T5 may be heterogeneous and may have different execution times. For example, as shown in Figure 10, the task T1 is longer than other tasks, and as a result, the VMs which tasks T2 to T5 are allocated to them should be idle after executing their task. Using the DVFS technique, we can mitigate the frequency of the VM1 to VM5's CPU to reduce their execution speed and energy consumption while meeting the deadline determined by the task T1. Also, the deadlines may be specified by the user or by the scheduling algorithm itself. The primary goal of DVFS-based scheduling is to detect the least possible frequency for each task, regarding their deadline can be met. In this case, the least power is consumed for tasks and workflow executions.

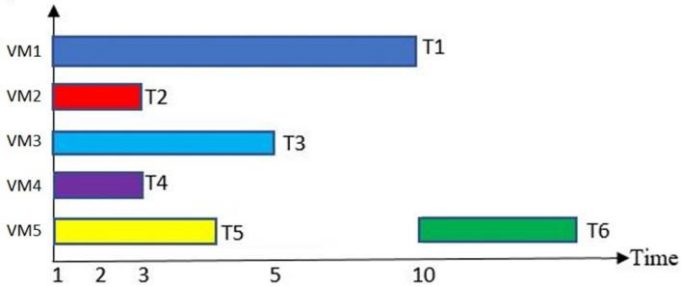


Figure 10: Gant chart of some tasks in a workflow

In this scheme, it is assumed that power consumption consists of static and dynamic and energy consumptions. $Energy_{dynamic}$ and $Energy_{static}$. Typically, in our scheduling approach, static energy consumption is ignored since dynamic energy consumption is more time-consuming and expensive.

$$Energy = Energy_{\text{stat}} + Energy_{\text{dynam}} \tag{26}$$

We compute $Power_{\text{dynam}}$ or dynamic power consumption as follows:

$$Power_{\text{dynam}} = K.v^2_s f \tag{27}$$

In which K is the constant of dynamic power consumption and is related to the capacities of the devices Also, v^2_s is the voltage of the s[th] level, and the j[th] VM, and f is the j[th] VM frequency. By using this equation, the energy consumption of the VMs can be computed as follows:

$$E_{\text{busy}} = \sum_{=1}^{n} K * v^2_s * f_s * ET = \sum_{=1}^{n} Power_{\text{dynam}} * ET \tag{28}$$

In which, ET is the execution time of the i[th] task on the jth VM, and v_s indicates that the i[th] is task is used on the jth VM, with the s[th] voltage level. Moreover, f_s is the frequency of processor j[th] VM, at the s[th] voltage level. Also, in the idle periods of VMs, their voltage has to be set to the lowest level to save the most energy. The energy consumption of idle periods for all available processors can be defined as follows:

$$E_{\text{ldle}} = \sum_{=1}^{p} k * v^2_{mn} * f_{mn} * IT = \sum_{=1}^{p} Power_{\text{dle}} * IT \tag{29}$$

In which v_{mn} and f_{mn} are the minimum voltage and frequency of the j[th] VM, and IT is considered to be the idle time of the j[th] VM. According to these equations, the total energy consumption required for DAG scheduling in a fog computing environment can be computed as follows:

$$E_{\text{total}} = E_{\text{Busy}} + E_{\text{ldle}} \tag{30}$$

4.2. Problem formulation

This section provides a formal definition of the considered workflow model and the energy model. For this purpose, we assumed that fog computing contains a collection of computational resources in the form of VMs, denoted by $VM=\{VM_1, VM_2, VM_3, ...\}$. The VMs can also work at different levels, such *as* $F\_DVFS=\{ F\_DVFS_1, F\_DVFS_2, F\_DVFS_3, ...\}$. F_DVFS_i indicates the i[th] setting for the fog processor's frequency and voltage and $F\_DVFS_i=(Frequency_i, Voltage_i)$. Also, we assumed that $F\_DVFS_i<F\_DVFS_j$ where i<j, or to be more specific, $Frequency_i<Frequency_j$ and $Voltage_i< Voltage_j$.

Besides, we deal with the workflows submitted to the fog environment as a directed acyclic graph (DAG). We indicate the set of workflows submitted to the fog by $W=\{W_1, W_2, W_3, ...,W_n\}$, in which each workflow contains some tasks, $W_i=\{T_1, T_2, T_3, ...\}$. Moreover, each workflow is considered a DAG, in which each node represents a task, and the edges specify data or control dependencies between tasks, in which E_{ij} defines the edge between the T_i and T_j, when $T_i \neq T_j$. Control dependency among tasks indicates that child tasks can be executed after all its parent tasks have fully executed, and their output data have been sent to it. Control dependencies only transfer the configuration parameters needed to execute the child task and transfer fewer data than data dependencies. However, the transferred data in the data dependencies are used as input data to the child process. In this scheme, the set of all direct predecessors of each workflow task can be computed as follows:

$$Predecessor(T_i) = \{ T_j \mid T_j, T_i \in E\} \tag{31}$$

Thus, regarding the entry task or tasks, their predecessor set should be empty, $Predecessor(T_{entry})=\{\}$. Furthermore, the set of all direct successors of each task can be computed as follows:

$$Successor(Ti) = = \{ T_j \mid T_i, T_j \in E\} \tag{32}$$

Furthermore, their successor set will be empty for the exit task or tasks, *Successor(T_{exit})*={}. To compute the average computation time of the T_i, first Equation 33 indicates how $Exection\_Time\ T\ ,\ F\_DVFS_k$ or the execution time of the T_i in the k[th] DVFS level should be computed:

$$Exection\_Time\ T\ ,F\_DVFS_k = \frac{Task\ len\ T}{VM\ j, F\_DVFS_k} \quad (33)$$

In which, $VM\ j,\ F\_DVFS_k$ specifies the speed of the j[th] VM using kth DVFS level, and $Task\_len\ T$ specifies the length of the task in terms of the millions of instructions per second. Also, the average time to execute the task T_i on the j[th] VM can be computed as follows:

$$Ave\ Exection\_Time\ T\ ,VMj = \frac{1}{Ndvfs}\sum_{k=1}^{Ndvfs} Exection\_Time\ T\ ,F\_DVFS_k \quad (34)$$

In which, *Ndvfs* is the number of DVFS levels in the VM. Furthermore, the average execution time of the task T_i on all VMs can be computed as follows:

$$Ave\ Exection\_Time\ T = \frac{1}{Nvm}\sum_{=1}^{Nvm} Ave(Exection\_Time\ T\ ,VM\ ) \quad (35)$$

In this scheme, the earliest start time of each task can be computed as follows:

$$STime\ T\ ,VM = \begin{cases} 0 & IF\ T_1\ is\ an\ entry\ task \\ \max\{avail(VM\ ), \max FT\ Tj\ + Com\_Time(T\ ,T\ ) & otherwise\ For\ each\ Tj \in Predecessor\ T \end{cases} \quad (36)$$

Where $avail(VM\ )$ is the time which j[th] VM becomes available to execute the requested task. In this scheme, the finish time of each task can be computed as follows:

$$FT(T\ ,VM\ ) = \begin{cases} deadline\ w_i & if\ \ T\ \ is\ an\ exit\ task \\ STime\ T\ ,VM\ + Ave\ \ Exection\_Time\ T & otherwise \end{cases} \quad (37)$$

In which, $deadline_{wi}$ denotes the deadline of the i[th] workflow. Also, the communication time of the data transfer between the T_i and T_j can be computed as follows:

$$Com\_Time\ T\ ,T\ = \begin{cases} 0 & IF\ \ VM\ T\ = VM(T\ ) \\ \dfrac{Data(T\ ,T\ )}{\lfloor Bandwidth\ VM\ \ Ti\ ,VM\ \ Tj} & otherwise \end{cases} \quad (38)$$

In which, $Bandwidth\ \ VM\ \ T\ \ ,VM(T\ )$ is the bandwidth between two VMs which execute the T_i and T_j tasks, and $Data(T\ ,\ T\ )$ denotes the amount of data that should be transferred between these tasks. Moreover, as shown in Figure 10, the makespan of the workflow w_i can be computed as follows:

$$makespan(w_i)=\{max\ \ FT\ T\ \ \ |\ T\ \ \in\ w_i\} \quad (39)$$

Generally, the scheduling algorithms may determine each task's following items: VM number, type of VM, DVFS setting of VM, the fog environment, which should execute the task, task order, etc. Figure 11 indicates a sample encoding in our proposed workflow scheduling scheme. As shown in this figure, each solution can be represented by a two-dimensional array in which for each task, the VM, which will execute the task, and the VM's DVFS level, should be determined. This scheme assumes that each VM can be tuned to a limited set of the DVFS levels. As a result, both the VM's number and DVFS levels are discrete numbers and adjusted by the proposed DMFO algorithm.

Task1	Task2	Task3	...	Taskn
VM1	VM3	VM5	...	VM2
DVFS Level1	DVFS Level2	DVFS Level1	...	DVFS Level3

Figure 11: A sample encoding in the proposed scheduling scheme

Equation 40 indicates the fitness function applied in this fog scheduling scheme:

$$Fitness = \begin{cases} 1 & if\ makespan\ w_i\ ?: deadline \\ \alpha * \frac{\text{makespan } w_i}{deadline\ w_i} + \beta * \frac{\text{Nwvm}}{\text{Nvm}} + \gamma * \frac{\text{S ommun at on}}{\text{W ommun at on}} & if\ makespan\ w_i < deadline \end{cases} \quad (40)$$

$$\alpha+\beta+\gamma = 1$$

In this equation, the *Scommunication* indicates the amount of the required amount of scheduling communications, and the *Wcommunication* exhibits the total amount of workflow communications and *Wcommunication*≥*Scommunication.* As shown in this equation, when dependent tasks are located in the same VM, their required communication cost is zero. Equations 41 and 42 indicate how the *scommunication* and wcommunication can be computed regarding the amount of data transfer between tasks and the amount of data transfer between VMs, respectively.

$$Wcommunication = \sum_{=1}^{N} \sum_{=1} comm\ T\ , sucessor(T\ ) \quad (41)$$

$$Scommunication = \sum_{=1}^{NVM} \sum_{=1} comm\ VM\ , VM_x \quad (42)$$

$$If\ VM_j = VM_X\ Then\ comm = 0$$

In which, L is the number of workflow levels, NT_i determines the number of tasks in the i[th] level of the workflow, and $comm\ T\ , sucessor(T\ )$ is the amount of communication between the T_j and its successor tasks. Furthermore, $comm\ VM\ , VM_x$ specifies the amount of communication between two VMs which execute two dependent tasks.

4.3. Finding task order

List-based scheduling methods first compute the workflow tasks' priorities in a DAG and rank them in non-increasing order. HEFT is one of the most popular list scheduling method which has been provided in the literature. For finding the order of task execution in the scientific workflows, we benefit from the task prioritization method supplied in the HEFT or heterogeneous earliest finish time algorithm. HEFT is a heuristic scheduling method for inter-dependent tasks onto a network of heterogeneous workers taking communication time into account. For inputs, HEFT takes a set of tasks, represented as a directed acyclic graph, a collection of VMs, the times to execute each worker, and the times to communicate each job's results between each pair of workers. It descends from list scheduling algorithms. HEFT algorithm first determines the priorities of the tasks and then assigns tasks to the workers. The rank of each task indicates its execution turn in the workflow scheduling. Thus, tasks with the lower rank will be executed first, and tasks with the higher ranks will be executed later and will have the lowest priority for execution. Equation 43 indicates the rank should be computed for each workflow task:

$$Rank\ Ti\ = Ave\ Exection\_Time\ Ti\ + \{\max(Com\_Time\ Ti, Tj\ + Rank\ Ti\ \ | Tj \in \text{Successor Ti}\ \} \quad (43)$$

Where T_i is the i[th] task in the workflow and $Ave\ Exection\_Time\ T_i$ is the average execution cost of the i[th] task. As outlined before, Successor T_i specifies the successor tasks of the T_i, and $Com\_Time\ T_i, T_j$ specifies the communication cost between the T_i and T_j.

4.4. DMFO-DE based workflow scheduling

The pseudo-code of the DVFS-based workflow scheduling using DMFO-DE is shown in Figure 12. As shown in this algorithm, first, the required parameters for running the algorithm are tuned. Then, the task prioritization method of the HEFT algorithm is used to find the order of the task execution in the workflow, and then the

DMFO-DE algorithm is used to find the best possible location for the tasks. Then, tasks are allocated to the required VM for execution by the order and setting specified in the best solution.

5. Results

This section presents the results of the experiments conducted to evaluate the performance of the proposed scheduling framework. There are several interesting simulation frameworks for fog computing, such as iFogSim and EdgeCloudSim. In this scheduling scheme, we have used the iFogSim simulator to conduct the required simulations, an efficient open-source tool for modeling and simulating resource management in IoT[61] and fog/edge computing networks. The iFogSim simulator works with the CloudSim, another open-source java-based simulator simulating cloud computing environments and managing its resources. The iFogSim simulator applies the CloudSim to deal with the events among fog computing components. In CloudSim, various entities such as data centers use message passing for communication. The proposed workflow scheduling algorithm is evaluated on the scientific workflows such as Epigenomics, CyberShake, LIGO, SIPHT, and Montage. Figure 13 shows the general structure of the employed scientific workflows.

Procedure Fog_scheduling()

Input: Scientific Workflow Wi
DVFS levels
Deadline of Scientific Workflow Wi

Set the number of VMs and their features
Set the bandwidth among VMs

Read workflow data from its DAX file
Compute the workflow tasks' rank based on the HEFT ranking method
Sort tasks in increasing order of their rank

Compute the workflow slack time
Allocate the slack time to the workflow levels

Set the objective function

Use the DMFO-DE optimization algorithm to obtain the best solution

For each Ti task in workflow Wi
Find the VMi and DVFS_Leveli from the best solution
Find the predecessor set of Ti

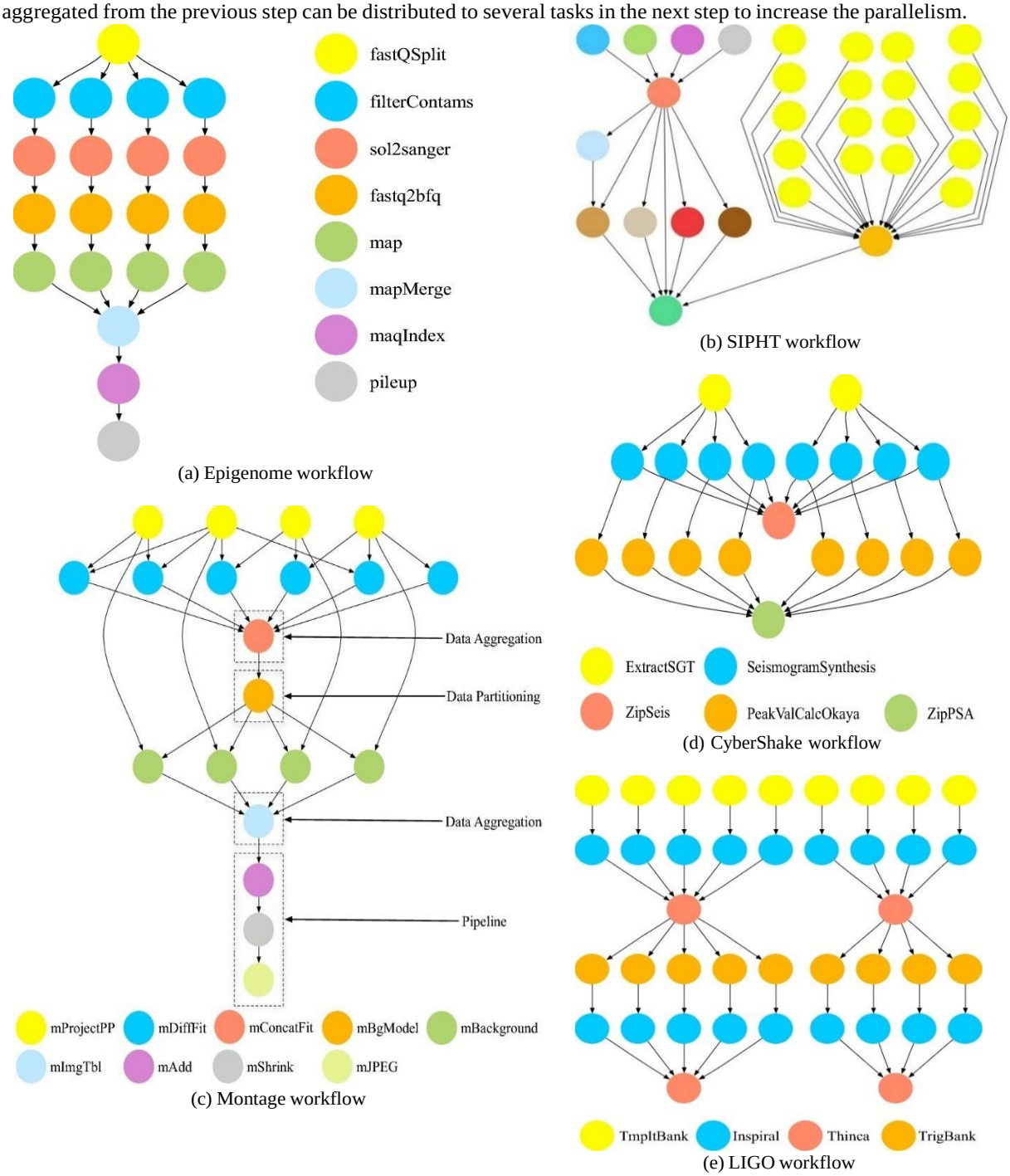
While(VMi is not idle **or** all Ti's predecessor are not executed **)**
Wait
End
Allocate the DVFS levels to the VMs
Execute task Ti on the VMi

End

Figure 12: DVFS-based workflow scheduling using DMFO-DE

Figure 13.a depicts the Epigenomics workflow's architecture, a data processing workflow that indicates the genome sequencing operations' execution and is applied by the Epigenome center. In this workflow, the DNA sequence data is produced by a genetic analysis system, split into many chunks that can be processed in parallel. Furthermore, each data chunk is converted to a format required by the sequence aligner. Then noisy sequences are filtered. Besides, a map to detect the density of sequence at every genome position is created.
Also, Figure 13.b exhibits the SIPHT workflow structure, a program that uses a workflow for automating the search for sRNA encoding-genes for all bacterial replicons[62]. It is created for a bioinformatics project at Harvard University in searching for untranslated RNAs for regulating processes like virulence or secretion in bacteria. Furthermore, Montage is an open-source toolkit for generating custom mosaics of the sky. It is presented as a workflow that can be run in various Grid, cloud, and even fog computing environments. Figure 13.c shows the general structure of the Montage workflow. Besides, Figure 13.d shows the structure of the LIGO workflow[63], which is applied to produce

and evaluate the gravitational waveforms for compact binary systems. In addition, Figure 13.e shows the general structure of the CyberShake workflow, which is typically employed by the earthquake center of southern California to analyze earthquake effects. The simplest structure in the scientific workflows is a task that processes input data to create an output. These tasks can then be combined sequentially to produce a pipeline structure in which each task input is the output of the previous task in the pipeline. Another structure is also denoted as data distribution or partitioning tasks that divide their output data into several chunks to be used by several tasks. As an advantage, such partitioning can lead to an increase in the parallelism level. On the other hand, data aggregation jobs aggregate several jobs' outputs and generate a combined data product. Another data structure is the data redistribution task. The data aggregated from the previous step can be distributed to several tasks in the next step to increase the parallelism.



(a) Epigenome workflow

(b) SIPHT workflow

(c) Montage workflow

(d) CyberShake workflow

(e) LIGO workflow

Figure 13: Applied scientific workflows

For evaluation of the scientific workflows, in [34], the authors provided a workflow generator tool that creates arbitrary size scientific workflows XML format that contains data about the task size and the amount of communication between dependent tasks. For conducting the required experiments and evaluating the proposed algorithm, as shown in Table 2, four synthetic Montage workflow with 30, 60, 100, and 1000 tasks are utilized to prepare their DAX. Regarding the Epigenome, four synthetic workflows with 24, 46, 100, and 997 tasks are used, which the workflow generator provides their DAX. Besides, four synthetic SIPHT workflows with 30, 60, 100, and 1000 tasks are utilized. Furthermore, the depicted results are averaged from 40 different runs of the investigated optimization algorithms. Regarding the Epigenome, four synthetic workflows with 24, 46, 100, and 997 tasks are used, which the workflow generator provides their DAX. Besides, four synthetic SIPHT workflows with 30, 60, 100, and 1000 tasks are utilized. Furthermore, the depicted results are averaged from 40 different runs of the investigated optimization algorithms.

5.1. Fitness values

Table 2 indicates the parameters considered in the experiments conducted to verify the proposed scheduling algorithm's effectiveness. As specified in Table 2, for all algorithms, 60 solutions are considered as their population, and the exhibited results are the average of 30 different runs of the studied algorithms. Besides, we use two deadlines, denoted as deadline1 and deadline2, in our experiments, which deadline 2 is more than deadline1. Table 3 lists the parameter setting applied for different optimization algorithms in the conducted experiments. The WOA or Whale Optimization Algorithm[64, 65] is proposed by Mirjalili et al. and is inspired by the humpback whales' hunting behavior. This algorithm can handle continuous optimization problems and benefits from humpback whales' actions such as searching the prey, bubbling, and encircling. Also, SCA or Sine-cosine algorithm is proposed by Mirjalili and can be considered as a swarm-based optimization algorithm, in which the optimization process is performed using the sine and cosine functions[66].

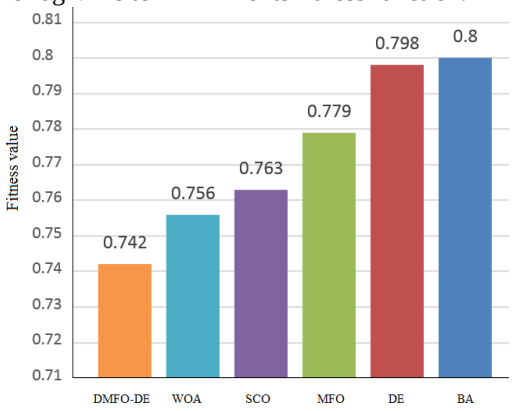
Table 2: Simulation parameters

Parameter		**Value**
Available VMs		80 VMs
Available bandwidth		100 Mbps, 1000 Mbps
VM's type		Homogeneous
Epigenomics		24 Nodes DAX, 46 Nodes DAX, 100 Nodes DAX, 997 Nodes DAX
SIPTH		30 Nodes DAX, 50 Nodes DAX, 100 Nodes DAX, 1000 Nodes DAX
Montage		25 Nodes DAX, 50 Nodes DAX, 100 Nodes DAX, 1000 Nodes DAX
Algorithm runs		30
Number of solutions		60 solutions
Deadline1		Makespan + 0.2* makespan
Deadline2		Makespan + 0.4* makespan
DVFS levels		6 levels
Level	**Voltage**	**Speed percentage**
1	1.5	100%
2	1.4	90%
3	1.3	80%
4	1.2	70%
5	1.1	60%
6	1.0	50%

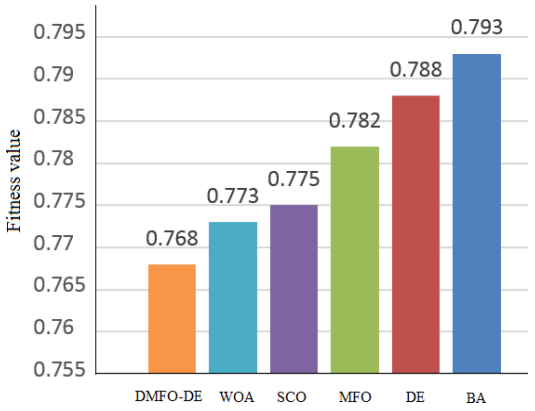
Table 3: The algorithms' parameters

Algorithm	**Parameter**	**Value**
Proposed DMFO-DE	α	0.4
	β	0.3
	γ	0.3
WOA	$r_1, r_2, and\ p$	rand
	a	2 to 0
	C	2 X r_2

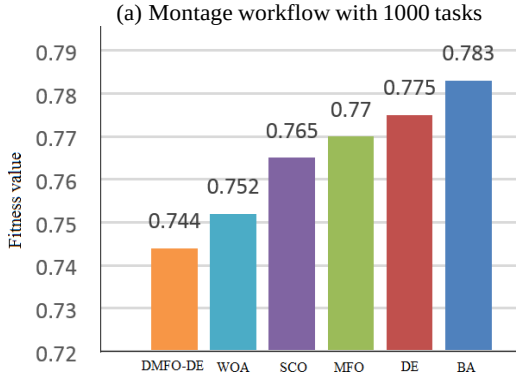
Figure 14 indicates the investigated optimization algorithms' fitness values achieved from scheduling workflow Montage, LIGO, SIPHT, and Cybershake with 1000 tasks, as well as Epigenomics workflow with 997 tasks. Since we have considered 80 VMs for the fog computing environment in these experiments, and the scheduling schemes apply most of these VMs for scheduling the tasks in each workflow level, the fitness value will be close to one. However, regarding the results shown by the various parts of Figure 13, our scheme can achieve the lowest fitness value among several optimization algorithms for different optimization algorithms. On the other hand, since our proposed DMFO-DE is a discrete algorithm and benefits from the OBL method, it can properly allocate the tasks to the fog VMs to minimize its fitness function.



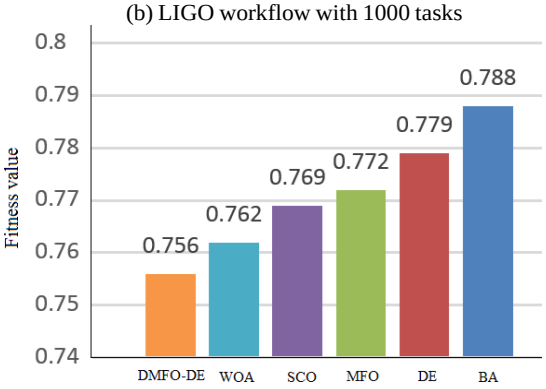
(a) Montage workflow with 1000 tasks



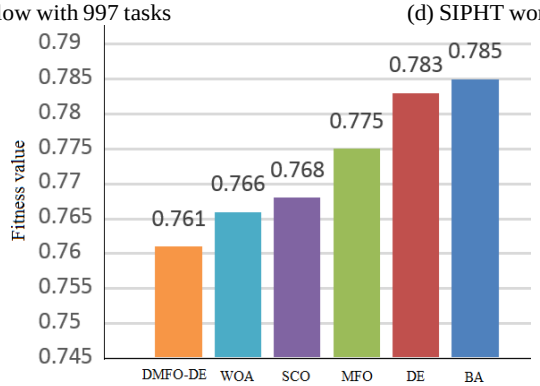
(b) LIGO workflow with 1000 tasks



(c) Epigenomics workflow with 997 tasks



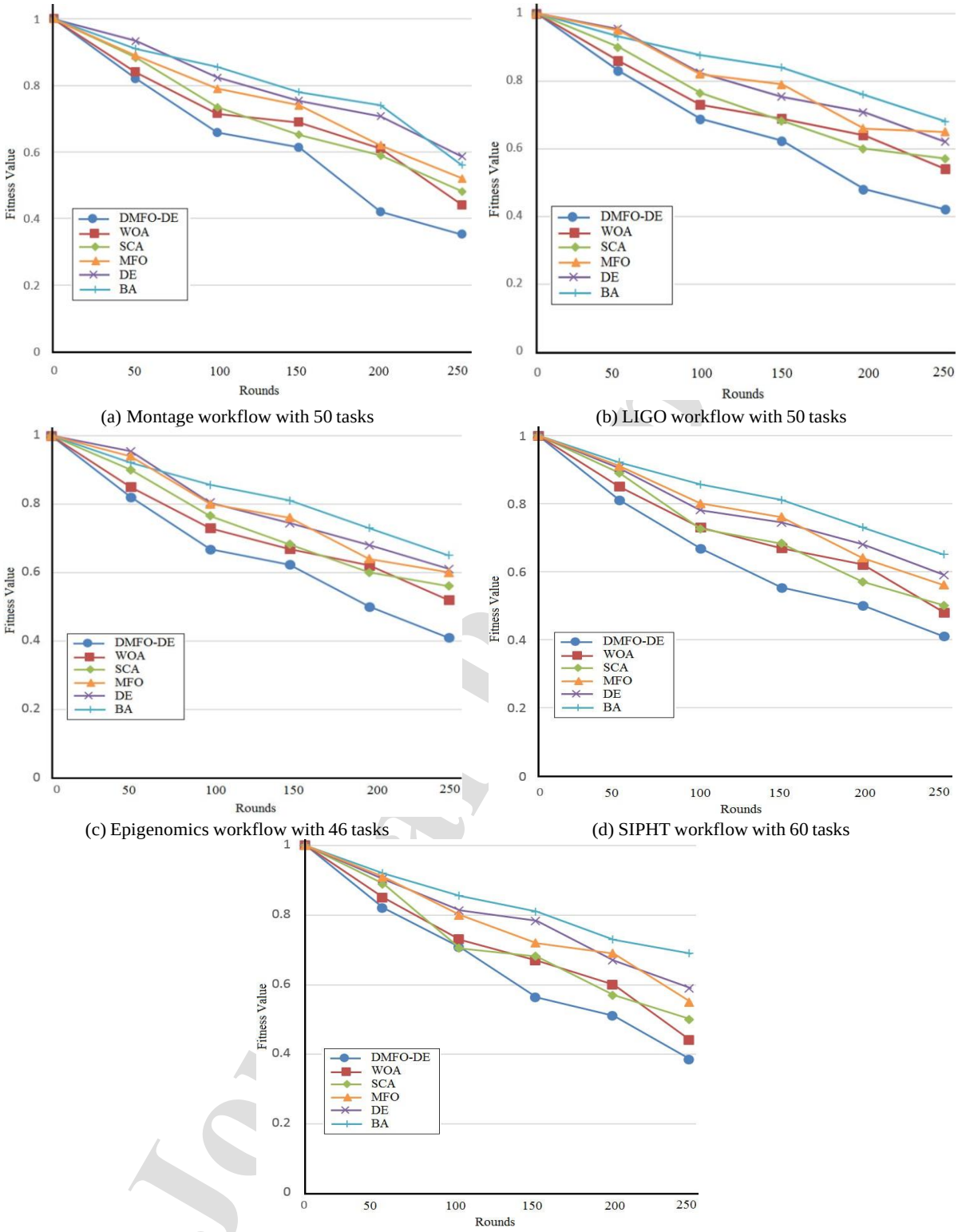
(d) SIPHT workflow with 1000 tasks



(e) CyberShake workflow with 1000 tasks

Figure 14: Fitness values for workflows with 1000 tasks

Figure 15 indicates the investigated optimization algorithms' investigated fitness values applied to schedule Montage, LIGO, and Cybershake workflows with 50 tasks, SIPHT workflows with 60 tasks, and Epigenomics workflow with 46 tasks. Again, in all studied algorithms, 60 solutions are used, and 30 different runs are averaged.



(a) Montage workflow with 50 tasks

(b) LIGO workflow with 50 tasks

(c) Epigenomics workflow with 46 tasks

(d) SIPHT workflow with 60 tasks

(e) CyberShake workflow with 50 tasks

Figure 15: Fitness values for workflows with 50 tasks

As shown in this figure, DMFO-DE can rapidly converge to the near-optimal solution and achieves better results than other studied investigation algorithms. Since we have used 80 VMs in our simulations and these workflows use fewer

VMs, and as a result, the fitness value of these experiments will be less than the fitness of workflow scheduling with 1000 tasks. At first, the solution has one's fitness in all algorithms because the randomly created solutions provide schedules that cannot satisfy the specified deadline. However, after the algorithms are executed, the optimization algorithms can find better schedules and mitigate the fitness value, in which our scheme provides better results.

5.2. Energy consumption improvements

This subsection presents the improvements which have been achieved in the energy consumption context. In these experiments, we employed Equation 44 for computing the percentage of the total energy consumption improvements.

$$Percentage\ of\ the\ energy\ consumption\ improvement = \frac{Energy\ usage\ of\ the\ workflow\ scheduling \quad Energy\ usage\ of\ the\ DVFS\ based\ workflow\ scheduling}{Energy\ consumption\ of\ the\ workflow\ scheduling} * 100 \tag{44}$$

In this equation, we first compute the energy consumption of the achieved scheduling for workflow, and then we deduce it by the energy consumption of the DVFS-based workflow scheduling. Figures 16 to 25 depict the experiments' results on the scientific workflows. Figure 16 indicates the energy reduction percentage in Montage workflows with 25 and 50 tasks, and Figure 17 exhibits the energy consumption reduction for the Montage workflows with 100 and 1000 tasks. It exhibits the scheduling results using the DMFO-DE, WOA, SCA, MFO, DE, and BA algorithms. As can be concluded from this figure, our approaches can better mitigate the energy required to schedule different-sized Montage workflows and outperform other algorithms.

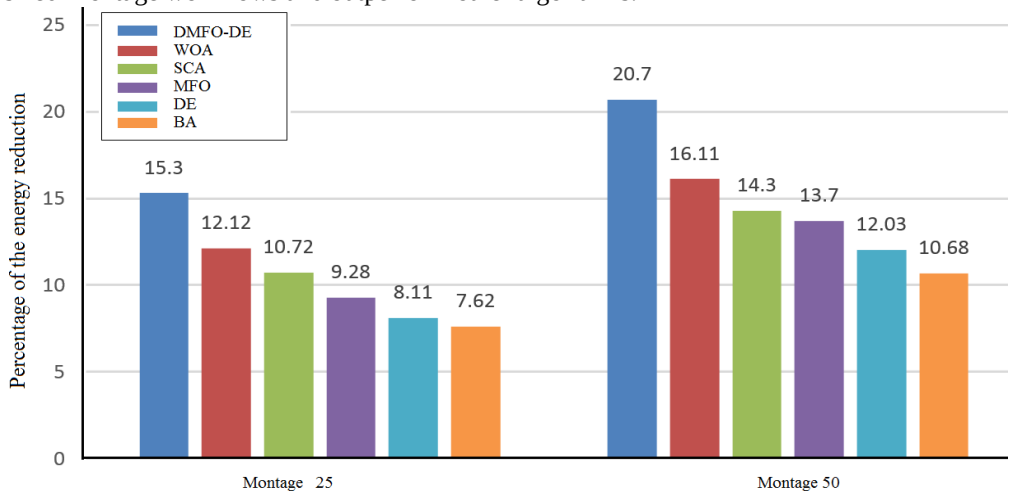


Figure 16: Energy consumption for Montage workflows with 30 and 50 tasks

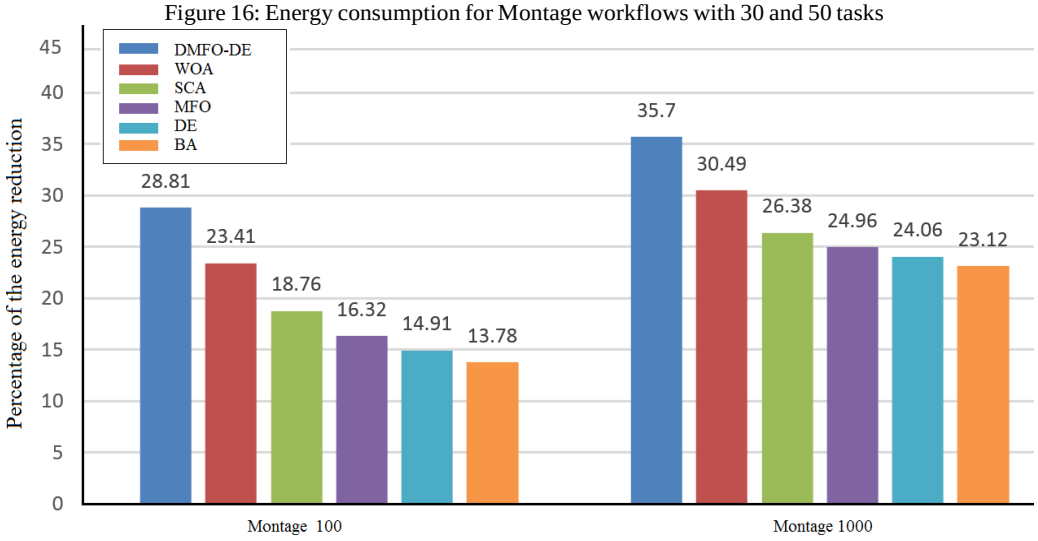


Figure 17: Energy consumption for Montage workflows with 100 and 1000 tasks

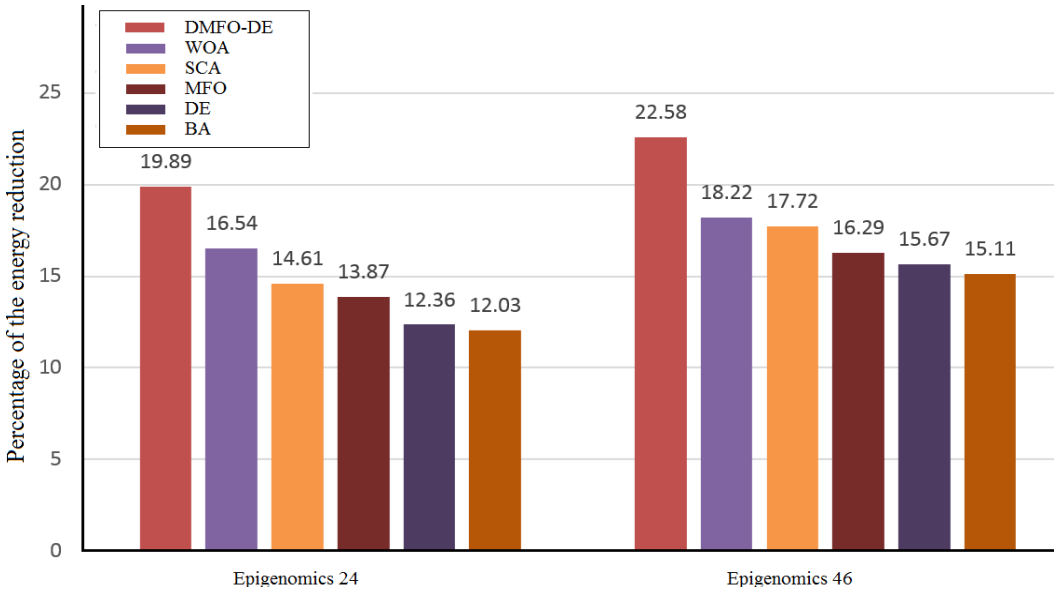


Figure 18: Energy consumption for Epigenomics workflows with 24 and 46 tasks

Figure 18 indicates the percentage of the energy reduction in two Epigenomics workflows with 24 and 46 tasks. It exhibits the scheduling results using the DMFO-DE, WOA, SCA, MFO, DE, and BA algorithms. As can be concluded from this figure, our approaches can better mitigate the energy required to schedule different-sized Montage workflows and outperform other algorithms. Figure 19 indicates the percentage of the energy reduction in the 4th scenario, in which two Epigenomics workflows with 100 and 997 tasks are used. It exhibits the scheduling results using the DMFO-DE, WOA, SCA, MFO, DE, and BA algorithms without DVFS. As can be concluded from this figure, our approaches can minimize the energy required to schedule different size Montage workflows.

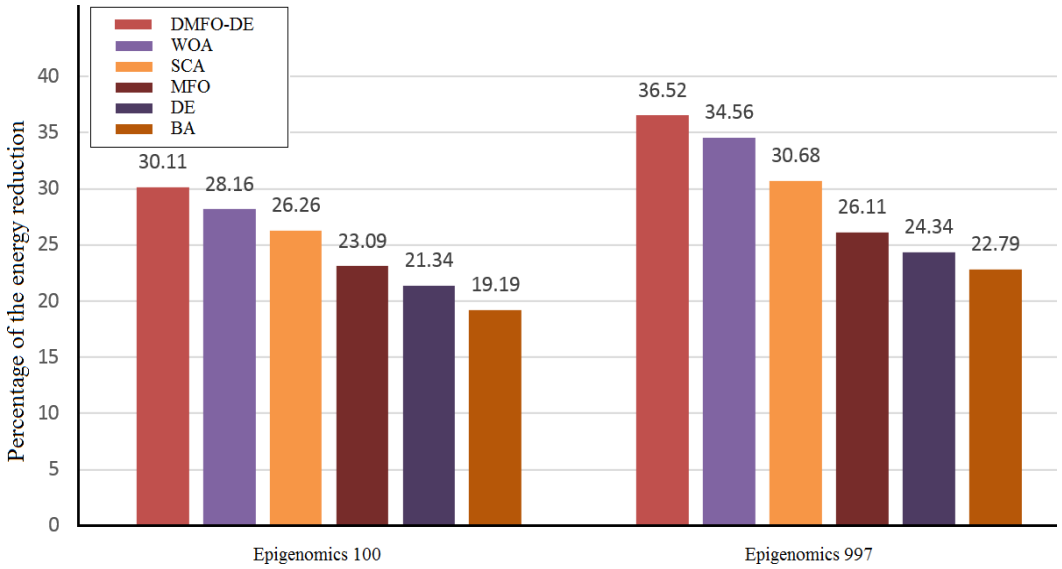


Figure 19: Energy consumption for Epigenomics workflows with 100 and 997 tasks

Figure 20 indicates the percentage of the energy consumption reduction when two LIGO scientific workflows with 30 and 50 tasks are applied. This figure exhibits the scheduling results using the DMFO-DE, WOA, SCA, MFO, DE, and BA algorithms. As shown in this figure, our approaches can better mitigate the energy required to schedule different-sized Montage workflows and outperform other algorithms.

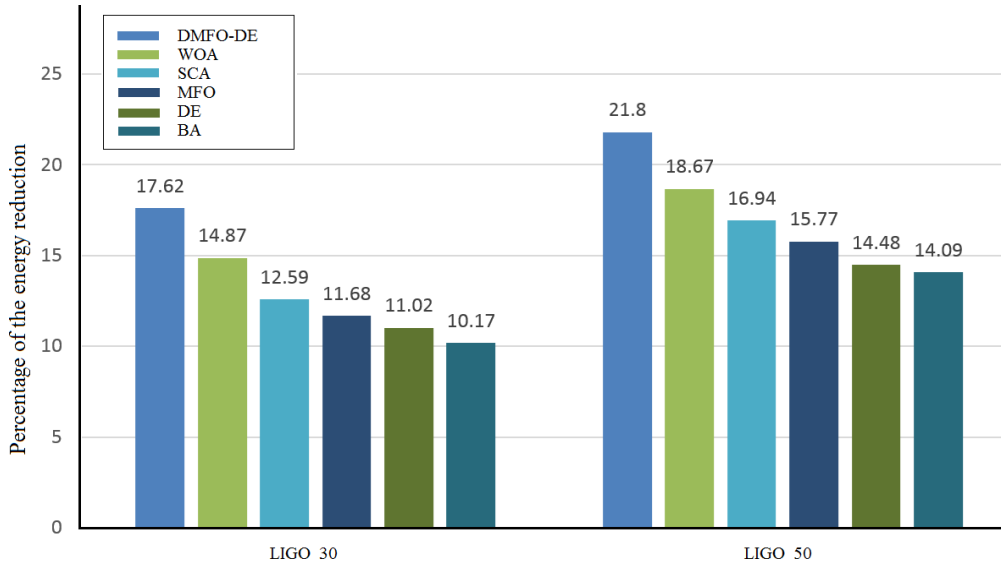


Figure 20: Energy consumption for LIGO workflows with 30 and 50 tasks

Figure 21 indicates the percentage of the energy reduction when two LIGO workflows with 100 and 1000 tasks are used. The experimental results exhibit that our workflow scheduling approach using DMFO-DE can decrease the scheduling power consumption for different-sized Montage workflows and outperform the other algorithms.

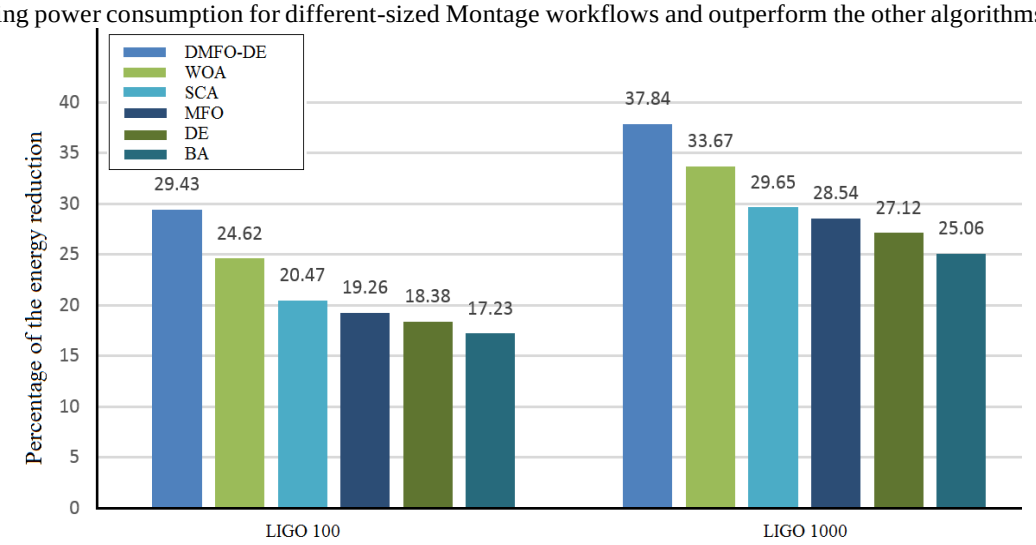


Figure 21: Energy consumption for LIGO workflows with 100 and 1000 tasks

Figure 22 exhibits the percentage of energy consumption mitigation for workflow scheduling using SIPHT workflows with 30 and 50 tasks. Figure 23 indicates the results of the same experiment for SIPHT workflows with 100 and 1000 tasks. Furthermore, Figures Figure 24 and 25 indicate the energy consumption experiment for the CyberShake with four different sizes. As shown in all these figures, our proposed optimization algorithm achieves better results than other algorithms.

At last, from the conducted experiments, it can be concluded that the following items are of the essential factors in energy consumption reduction in DVFS-based scheduling:

- Workflow scheduling deadline.
- Number of VMs applied in the scheduling.
- The number of workflow tasks.
- The number of tasks that can be executed simultaneously in each level of the scientific workflows.
- The deadline distribution policy is applied to distribute the slack time among different workflow levels.

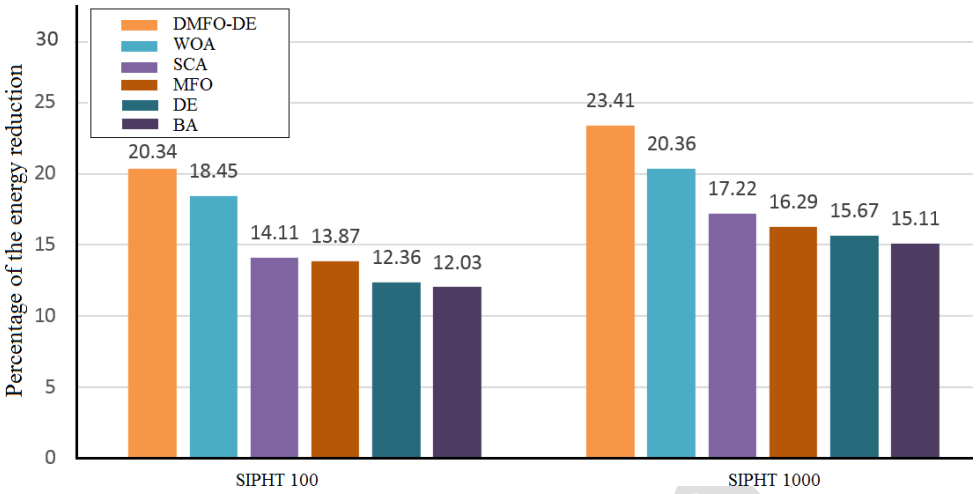


Figure 22: Energy consumption for SIPHT workflows with 30 and 50 tasks

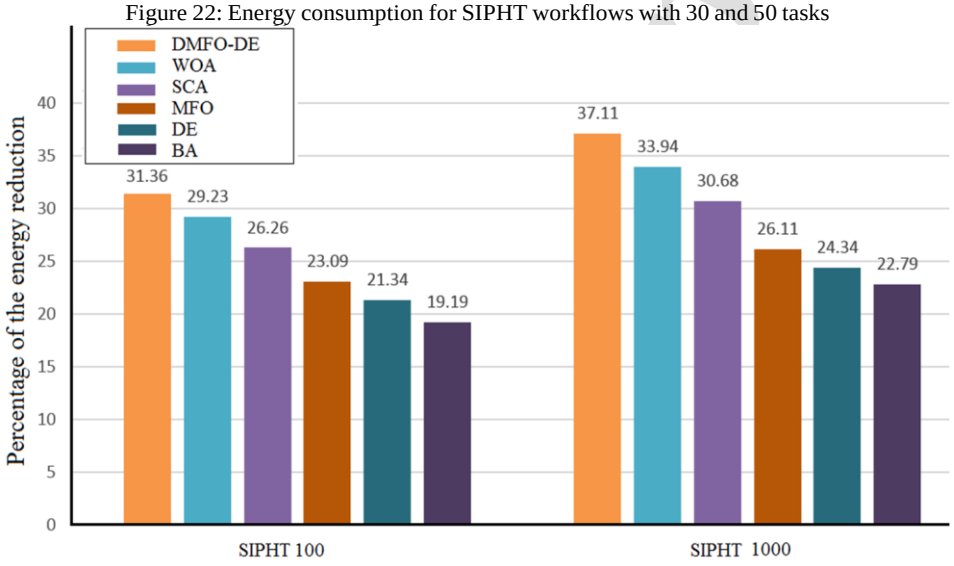


Figure 23: Energy consumption for SIPHT workflows with 100 and 1000 tasks

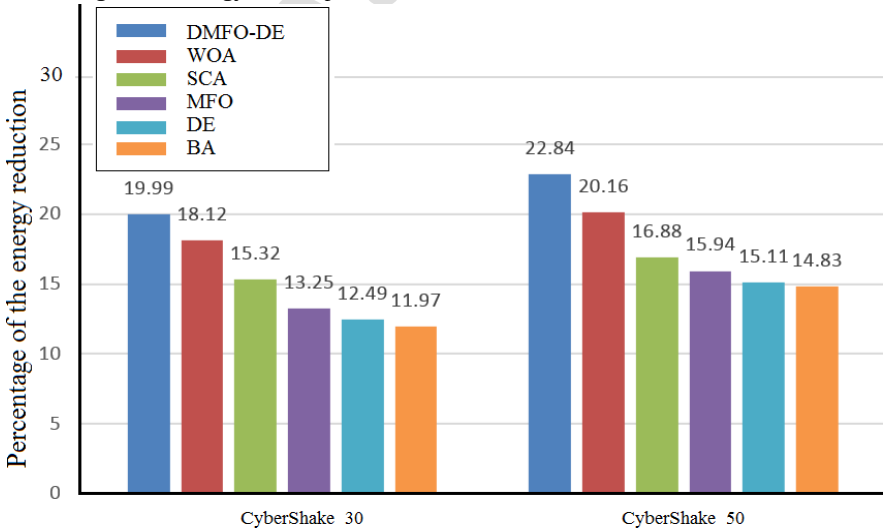


Figure 24: Energy consumption for CyberShake workflows with 30 and 50 tasks

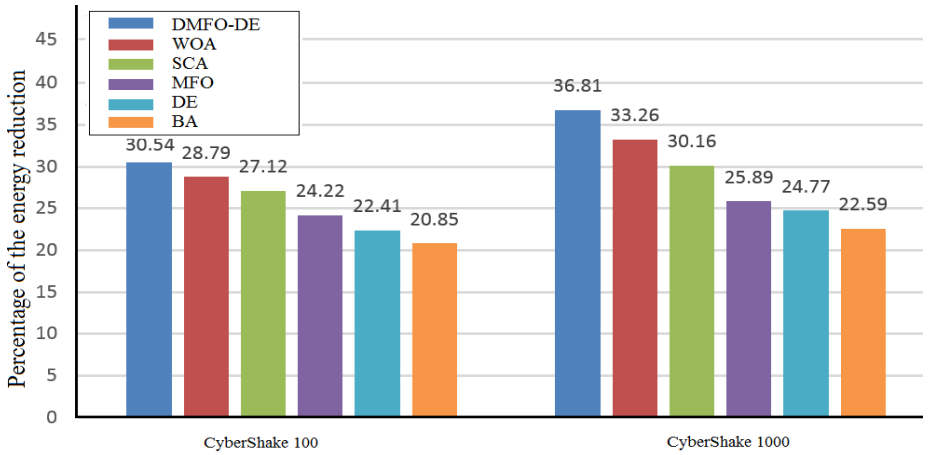


Figure 25: Energy consumption for CyberShake workflows with 100 and 1000 tasks

Table 4: Percentage of energy consumption improvement for scheduling with three DVFS levels with deadline1									
Workflow	**Size**	**Value**	**DMFO-DE**	**DMFO**	**WOA**	**SCA**	**MFO**	**DE**	**BA**
LIGO	1000	Ave:	40.12	39.01	36.49	32.53	30.27	29.33	27.81
		Best:	40.86	39.97	37.07	33.12	31.03	30.15	28.59
LIGO	100	Ave:	33.48	30.25	27.61	23.52	21.72	20.19	19.11
		Best:	34.12	31.08	28.31	24.26	22.34	20.98	19.87
LIGO	50	Ave:	23.91	21.28	20.62	18.11	16.57	15.93	15.32
		Best:	24.75	21.89	21.38	18.96	17.72	16.74	16.14
LIGO	30	Ave:	19.83	17.05	16.58	14.674	13.29	12.95	11.27
		Best:	20.71	17.92	17.19	15.11	14.08	13.67	11.84
Epigenomics	997	Ave:	39.87	38.14	37.62	33.95	29.61	26.24	24.88
		Best:	40.52	38.96	38.30	34.72	30.35	26.99	25.51
Epigenomics	100	Ave:	33.84	31.97	31.65	29.53	26.47	23.22	21.53
		Best:	34.48	32.58	32.30	30.19	27.13	23.96	22.65
Epigenomics	46	Ave:	22.29	21.34	20.64	19.62	17.69	16.44	16.28
		Best:	22.92	21.99	21.27	20.25	18.21	17.05	16.93
Epigenomics	24	Ave:	21.53	18.66	18.43	16.86	14.76	13.69	13.14
		Best:	22.37	19.27	19.16	17.38	15.49	14.43	13.75
SIPHT	1000	Ave:	39.27	36.46	35.61	33.60	28.43	26.22	24.51
		Best:	40.15	37.08	36.24	34.29	29.33	26.95	25.10
SIPHT	100	Ave:	34.65	33.23	32.58	29.86	25.43	23.54	22.39
		Best:	35.13	33.94	33.14	30.57	26.12	24.26	23.07
SIPHT	60	Ave:	25.75	22.98	22.36	19.45	17.94	17.12	16.81
		Best:	26.58	23.64	22.83	20.13	18.56	17.73	18.54
SIPHT	30	Ave:	22.87	21.03	20.42	17.12	15.86	13.98	13.11
		Best:	23.51	21.85	20.95	17.78	16.40	14.59	13.72
CyberShake	1000	Ave:	39.84	37.16	36.52	33.94	27.68	26.81	24.38
		Best:	40.49	37.82	37.08	34.51	28.54	27.44	24.93
CyberShake	100	Ave:	33.76	32.39	31.67	29.59	26.14	24.28	22.92
		Best:	34.42	32.88	32.19	30.60	26.77	24.89	23.58
CyberShake	50	Ave:	25.69	24.33	23.53	19.82	18.31	17.36	16.92
		Best:	26.38	24.92	23.96	20.52	18.88	17.95	17.64
CyberShake	30	Ave:	23.57	22.42	21.38	17.64	15.28	14.93	13.47
		Best:	24.23	22.90	21.82	18.25	15.84	15.66	14.05
Montage	1000	Ave:	38.51	34.26	33.64	29.57	27.85	26.37	24.05
		Best:	39.27	35.05	34.15	30.14	28.38	27.92	24.66
Montage	100	Ave:	31.59	28.51	26.73	21.94	19.46	16.86	15.68
		Best:	32.16	29.16	27.48	22.62	20.02	17.49	16.37
Montage	50	Ave:	22.81	20.28	18.46	16.58	14.69	13.53	12.27
		Best:	23.49	21.06	19.08	17.12	15.46	14.07	12.82
Montage	25	Ave:	17.68	16.07	14.92	12.52	11.33	10.54	9.16
		Best:	18.48	16.90	15.59	13.27	11.85	11.15	9.73

Table 4 indicates the energy consumption reduction results for conducting workflow scheduling when three DVFS levels are incorporated with deadline2. Table 5 gives the results of the experiments conducted for workflow scheduling with six DVFS levels with deadline2. As shown in this table, our proposed discrete and optimization algorithm can outperform other algorithms regarding energy consumption reduction. Besides, Table 6 presents the same experiment results when more deadline is considered for the scheduling process.

Table 5: Energy consumption ratio for workflow scheduling with six DVFS levels with deadline2

Workflow	Size	Value	DMFO-DE	DMFO	WOA	SCA	MFO	DE	BA
LIGO	1000	Ave:	43.81	42.09	39.24	35.23	33.41	32.19	30.58
		Best:	44.33	42.73	39.61	35.79	33.84	32.68	30.92
LIGO	100	Ave:	36.23	33.39	29.87	25.74	24.16	22.98	21.86
		Best:	36.85	33.90	30.22	26.13	24.85	23.42	22.35
LIGO	50	Ave:	26.44	25.28	23.87	20.46	18.42	17.64	17.05
		Best:	26.83	25.96	24.21	20.82	18.91	18.07	17.48
LIGO	30	Ave:	21.48	19.24	18.67	16.58	15.92	14.33	13.29
		Best:	21.83	19.81	19.10	16.95	16.45	14.75	13.72
Epigenomics	997	Ave:	43.51	42.49	40.75	36.94	34.50	29.37	26.82
		Best:	43.98	43.03	41.14	37.51	34.96	29.81	27.33
Epigenomics	100	Ave:	36.62	34.10	33.71	32.26	30.15	26.54	24.28
		Best:	37.20	34.62	34.12	32.69	30.67	26.96	24.73
Epigenomics	46	Ave:	24.81	23.67	22.83	22.14	20.16	19.34	19.37
		Best:	25.29	24.13	23.37	22.58	20.73	19.67	19.81
Epigenomics	24	Ave:	23.13	21.68	20.17	18.42	16.76	15.46	16.75
		Best:	23.66	22.29	20.59	18.95	17.30	15.92	17.28
SIPHT	1000	Ave:	42.59	40.71	38.69	37.10	32.84	30.24	27.81
		Best:	43.02	41.23	39.14	37.59	33.22	30.79	28.19
SIPHT	100	Ave:	37.82	36.18	35.87	32.86	28.97	26.67	25.35
		Best:	38.27	36.84	36.25	33.31	29.37	27.11	25.81
SIPHT	60	Ave:	27.63	25.05	24.39	23.13	20.61	19.76	20.41
		Best:	28.11	25.77	24.92	23.69	21.04	20.14	20.83
SIPHT	30	Ave:	24.33	23.28	22.13	19.66	17.80	16.22	16.67
		Best:	24.69	23.87	22.64	20.15	18.29	16.75	17.29
CyberShake	1000	Ave:	42.49	41.09	39.56	37.23	31.86	29.57	28.90
		Best:	42.83	41.75	40.12	37.71	32.25	29.92	29.45
CyberShake	100	Ave:	36.94	35.39	34.22	32.49	29.34	27.58	25.63
		Best:	37.50	35.91	34.71	32.96	29.82	28.03	26.24
CyberShake	50	Ave:	27.62	26.16	25.84	23.31	21.68	20.34	19.34
		Best:	28.07	26.80	26.23	23.72	22.05	20.62	19.76
CyberShake	30	Ave:	25.53	23.98	23.73	20.58	18.92	16.82	16.27
		Best:	25.92	24.50	24.16	20.96	19.44	17.35	16.80
Montage	1000	Ave:	41.38	28.86	36.96	33.21	30.67	29.37	27.64
		Best:	41.85	29.38	37.43	33.70	31.05	29.82	28.21
Montage	100	Ave:	34.87	31.37	29.70	24.62	23.40	20.48	20.11
		Best:	35.38	31.82	30.18	24.99	23.83	20.89	20.56
Montage	50	Ave:	24.86	22.49	20.62	19.14	17.32	17.62	16.38
		Best:	25.17	22.97	21.05	19.62	17.84	18.13	16.89
Montage	25	Ave:	19.67	17.19	16.37	15.10	14.69	13.57	12.29
		Best:	20.10	17.74	16.82	15.58	17.12	13.91	12.74

5.3. Resource usage

This subsection indicates the improvements which have been achieved in the fog's virtual resource usage. A reduction in the number of applied virtual resources directly affects the execution costs of scientific workflows. Also, decreasing the number of VMs, reduce energy consumption, and can lead to a more green fog computing environment. Besides, using fewer VMs for each workflow, the throughput of the fog computing environment can be increased, and the fog can handle more workflows. Table 7 indicates the experiments' results regarding the ratio of resource usage for each workflow when deadline1 is used for scheduling. This table shows the experimental results for scientific workflows such as Epigenomics, SIPTH, LIGO, CyberShake, Epigenomics, and Montage scientific workflows. Also, for each

type of workflow, four different size DAXs are used in the experiments. These results are also achieved by computing the average of the 30 different executions of each optimization algorithm. As shown in this figure, our proposed DMFO-DE algorithm can outperform other optimization algorithms such as WOA, SCA, MFO, DE, and BA in reducing the number of VMs required to execute the workflow in a specific number of rounds.

Table 6: Energy consumption reduction for workflow scheduling with six DVFS levels and deadline2

Workflow	Size	Value	DMFO-DE	DMFO	WOA	SCA	MFO	DE	BA
LIGO	1000	Ave:	44.93	43.86	42.61	39.53	37.41	36.91	34.82
		Best:	45.70	44.37	43.12	40.87	38.86	37.39	35.26
LIGO	100	Ave:	37.35	35.46	32.47	29.14	28.66	26.28	23.61
		Best:	37..83	36.38	32.85	29.63	29.10	26.75	24.05
LIGO	50	Ave:	30.26	28.44	27.74	24.82	21.32	20.37	19.96
		Best:	30.63	29.03	28.26	25.36	21.74	20.74	20.52
LIGO	30	Ave:	25.81	23.32	22.77	20.48	18.25	17.88	17.32
		Best:	26.32	23.80	23.25	20.90	18.69	18.29	17.79
Epigenomics	997	Ave:	44.11	43.09	42.24	40.44	37.58	31.74	30.62
		Best:	44.57	43.62	42.87	40.89	38.08	32.20	31.25
Epigenomics	100	Ave:	39.32	38.26	37.21	36.16	36.12	30.11	27.72
		Best:	39.81	38.85	37.68	36.72	36.57	30.58	28.30
Epigenomics	46	Ave:	27.36	26.47	25.99	26.85	24.22	22.33	21.79
		Best:	27.84	26.93	26.53	27.46	24.73	22.79	22.15
Epigenomics	24	Ave:	26.80	25.72	24.32	22.28	19.15	18.71	18.15
		Best:	27.41	26.20	24.86	22.79	19.61	19.28	18.62
SIPHT	1000	Ave:	46.54	43.51	42.89	41.55	36.45	34.49	31.51
		Best:	46.97	44.05	43.47	41.97	36.93	34.92	31.98
SIPHT	100	Ave:	40.26	39.34	38.24	36.42	31.49	29.56	28.43
		Best:	40.73	39.76	38.76	36.91	31.92	30.04	28.94
SIPHT	60	Ave:	30.13	39.17	28.89	27.93	23.46	22.60	23.71
		Best:	30.67	39.72	29.47	28.47	23.89	23.08	24.12
SIPHT	30	Ave:	28.33	27.26	26.50	23.14	22.10	19.45	18.17
		Best:	28.96	27.81	27.13	23.68	22.68	19.93	18.59
CyberShake	1000	Ave:	44.11	43.35	42.25	40.82	35.16	33.51	31.29
		Best:	44.58	43.94	42.71	41.46	35.60	33.97	31.84
CyberShake	100	Ave:	39.89	38.37	37.12	36.11	33.42	31.74	28.72
		Best:	40.35	38.94	37.64	36.71	33.87	32.17	29.08
CyberShake	50	Ave:	31.21	30.25	29.24	28.18	21.68	23.68	22.44
		Best:	31.67	30.86	29.72	28.83	22.25	33.12	22.91
CyberShake	30	Ave:	28.13	27.05	26.46	24.68	18.92	19.37	18.74
		Best:	28.72	27.69	26.82	25.11	19.43	19.82	19.23
Montage	1000	Ave:	45.68	42.64	40.23	37.17	30.67	33.58	31.94
		Best:	46.07	43.19	40.68	37.68	31.15	34.03	32.46
Montage	100	Ave:	38.17	34.89	33.37	28.25	23.40	23.63	22.47
		Best:	38.65	35.41	33.75	28.72	23.86	24.15	22.85
Montage	50	Ave:	27.26	25.38	24.96	23.48	17.32	20.84	19.18
		Best:	27.71	26.02	25.62	23.94	17.80	21.35	19.73
Montage	25	Ave:	22.44	21.24	20.75	19.44	14.69	16.51	15.47
		Best:	22.87	21.83	21.34	19.89	15.25	16.98	15.88

Table 8 exhibits the resource usage ratio in the experiments in which deadline2 is used for workflow scheduling. As shown in this table, our proposed discrete and hybrid optimization algorithm gives better results than other optimization algorithms in different scientific workflows such as Epigenomics, SIPTH, LIGO, CyberShake, Epigenomics, and Montage. As shown in this table, our approaches can minimize the number of applied VMs in four different size workflows. Since our algorithm is discrete and can handle discrete solutions, it can better handle discrete problems such as scheduling, and it provides better results than the other algorithms. Also, by combining the MFO and DE, it prevents the local optima problem, and as shown in previous subsections, it can converge faster to the near-optimal results.

Table 7: Resource usage ratio in workflow scheduling with deadline1									
Workflow	**Size**	**Value**	**DMFO-DE**	**DMFO**	**WOA**	**SCA**	**MFO**	**DE**	**BA**
LIGO	1000	Ave:	0.87	0.89	0.90	0.91	0.92	0.94	0.97
		Best:	0.86	0.88	0.89	0.90	0.90	0.93	0.96
LIGO	100	Ave:	0.80	0.82	0.83	0.84	0.86	0.87	0.89
		Best:	0.78	0.81	0.81	0.81	0.85	0.85	0.88
LIGO	50	Ave:	0.71	0.74	0.75	0.76	0.79	0.81	0.85
		Best:	0.70	0.73	0.71	0.73	0.77	0.79	0.83
LIGO	30	Ave:	0.70	0.72	0.74	0.75	0.76	0.79	0.81
		Best:	0.68	0.71	0.72	0.73	0.75	0.77	0.80
Epigenomics	997	Ave:	0.86	0.88	0.89	0.91	0.91	0.92	0.95
		Best:	0.85	0.87	0.87	0.9	0.9	0.91	0.93
Epigenomics	100	Ave:	0.78	0.80	0.82	0.84	0.85	0.86	0.88
		Best:	0.76	0.79	0.80	0.82	0.84	0.85	0.86
Epigenomics	46	Ave:	0.74	0.76	0.80	0.81	0.82	0.84	0.85
		Best:	0.72	0.75	0.78	0.79	0.81	0.82	0.84
Epigenomics	24	Ave:	0.71	0.73	0.75	0.76	0.79	0.82	0.83
		Best:	0.69	0.72	0.73	0.74	0.78	0.81	0.81
SIPHT	1000	Ave:	0.86	0.88	0.90	0.92	0.94	0.95	0.96
		Best:	0.84	0.87	0.88	0.91	0.93	0.93	0.94
SIPHT	100	Ave:	0.76	0.80	0.82	0.84	0.85	0.88	0.90
		Best:	0.74	0.78	0.81	0.82	0.84	0.87	0.88
SIPHT	60	Ave:	0.74	0.77	0.79	0.81	0.82	0.84	0.87
		Best:	0.73	0.76	0.77	0.80	0.80	0.83	0.85
SIPHT	30	Ave:	0.72	0.74	0.76	0.77	0.80	0.82	0.83
		Best:	0.70	0.73	0.75	0.75	0.78	0.80	0.81
CyberShake	1000	Ave:	0.84	0.85	0.86	0.87	0.89	0.9	0.93
		Best:	0.82	0.84	0.84	0.85	0.87	0.89	0.92
CyberShake	100	Ave:	0.80	0.82	0.84	0.86	0.87	0.88	0.90
		Best:	0.78	0.81	0.82	0.85	0.85	0.87	0.89
CyberShake	50	Ave:	0.73	0.75	0.78	0.81	0.83	0.84	0.86
		Best:	0.72	0.74	0.76	0.80	0.82	0.82	0.85
CyberShake	30	Ave:	0.71	0.73	0.75	0.76	0.77	0.80	0.82
		Best:	0.70	0.72	0.73	0.74	0.76	0.79	0.81
Montage	1000	Ave:	0.85	0.87	0.89	0.90	0.92	0.93	0.94
		Best:	0.82	0.86	0.87	0.89	0.91	0.91	0.92
Montage	100	Ave:	0.77	0.80	0.83	0.85	0.86	0.88	0.89
		Best:	0.75	0.79	0.81	0.83	0.84	0.87	0.87
Montage	50	Ave:	0.75	0.78	0.82	0.83	0.84	0.86	0.87
		Best:	0.73	0.76	0.80	0.81	0.83	0.84	0.85
Montage	25	Ave:	0.71	0.73	0.74	0.77	0.78	0.83	0.84
		Best:	0.70	0.72	0.72	0.76	0.76	0.81	0.82

5.4. Overheads

Typically, the MFO algorithm executes the Quicksort algorithm two times in each iteration for sorting flames and their fitness values. Generally, in the average case, the Quicksort algorithm's performance is O(nlogn), and its worst-case performance is $O(n^2)$, which can reduce the performance of the optimization process. As an advantage, our proposed DMFO-DE algorithm incurs fewer processing overheads than the basic MFO algorithm because it needs less execution of the MFO and is aided by the DE algorithm. Thus, using DE helps to explore the problem space better and reduce the number of Quicksort executions. This subsection indicates the number of Quicksort algorithm's execution in the workflow scheduling process using MFO and the proposed DMFO-DE algorithms. Figure 26 shows the number of times which the sorting algorithm is executed in the MFO and the DMFO-DE algorithms to schedule the Montage workflows with 25 tasks in 150 rounds. As can be seen in this figure, the DMFO-DE algorithm, which in some rounds uses the DE algorithm, needs to execute the sorting algorithm fewer times, and this makes it much faster than the MFO.

Table 8: Resource usage ratio in workflow scheduling with deadline2									
Workflow	**Size**	**Value**	**DMFO-DE**	**DMFO**	**WOA**	**SCA**	**MFO**	**DE**	**BA**
LIGO	1000	Ave:	0.79	0.80	0.85	0.87	0.87	0.88	0.89
		Best:	0.77	0.79	0.83	0.86	0.85	0.87	0.88
LIGO	100	Ave:	0.72	0.76	0.78	0.80	0.81	0.83	0.85
		Best:	0.70	0.75	0.76	0.79	0.80	0.82	0.83
LIGO	50	Ave:	0.68	0.70	0.72	0.75	0.77	0.78	0.81
		Best:	0.67	0.71	0.71	0.74	0.75	0.77	0.79
LIGO	30	Ave:	0.64	0.68	0.73	0.74	0.76	0.77	0.79
		Best:	0.63	0.66	0.71	0.72	0.75	0.76	0.78
Epigenomics	997	Ave:	0.77	0.81	0.84	0.85	0.87	0.89	0.90
		Best:	0.76	0.79	0.82	0.84	0.86	0.88	0.88
Epigenomics	100	Ave:	0.74	0.77	0.81	0.82	0.83	0.82	0.84
		Best:	0.73	0.76	0.80	0.80	0.81	0.83	0.82
Epigenomics	46	Ave:	0.69	0.72	0.73	0.74	0.76	0.77	0.79
		Best:	0.67	0.71	0.72	0.73	0.75	0.76	0.78
Epigenomics	24	Ave:	0.63	0.67	0.71	0.72	0.75	0.76	0.76
		Best:	0.62	0.66	0.70	0.70	0.73	0.74	0.75
SIPHT	1000	Ave:	0.79	0.81	0.84	0.86	0.87	0.89	0.92
		Best:	0.77	0.80	0.83	0.84	0.86	0.88	0.90
SIPHT	100	Ave:	0.75	0.78	0.80	0.82	0.82	0.84	0.87
		Best:	0.73	0.77	0.79	0.80	0.81	0.83	0.85
SIPHT	60	Ave:	0.71	0.75	0.78	0.79	0.81	0.82	0.84
		Best:	0.70	0.74	0.76	0.77	0.80	0.81	0.83
SIPHT	30	Ave:	0.61	0.66	0.70	0.73	0.74	0.76	0.79
		Best:	0.60	0.64	0.68	0.72	0.72	0.75	0.77
CyberShake	1000	Ave:	0.78	0.81	0.83	0.84	0.86	0.87	0.89
		Best:	0.76	0.80	0.82	0.83	0.85	0.86	0.90
CyberShake	100	Ave:	0.72	0.73	0.75	0.78	0.81	0.83	0.84
		Best:	0.71	0.73	0.74	0.76	0.78	0.82	0.82
CyberShake	50	Ave:	0.67	0.68	0.71	0.75	0.76	0.79	0.81
		Best:	0.65	0.67	0.69	0.73	0.75	0.78	0.80
CyberShake	30	Ave:	0.61	0.66	0.67	0.71	0.74	0.76	0.77
		Best:	0.60	0.65	0.64	0.69	0.72	0.75	0.75
Montage	1000	Ave:	0.73	0.75	0.78	0.81	0.84	0.88	0.90
		Best:	0.71	0.74	0.76	0.79	0.82	0.86	0.88
Montage	100	Ave:	0.70	0.73	0.77	0.80	0.82	0.83	0.85
		Best:	0.68	0.72	0.76	0.78	0.81	0.80	0.83
Montage	50	Ave:	0.66	0.69	0.71	0.72	0.76	0.79	0.82
		Best:	0.64	0.68	0.70	0.70	0.74	0.77	0.81
Montage	25	Ave:	0.62	0.64	0.66	0.69	0.73	0.77	0.80
		Best:	0.61	0.63	0.64	0.68	0.71	0.76	0.78

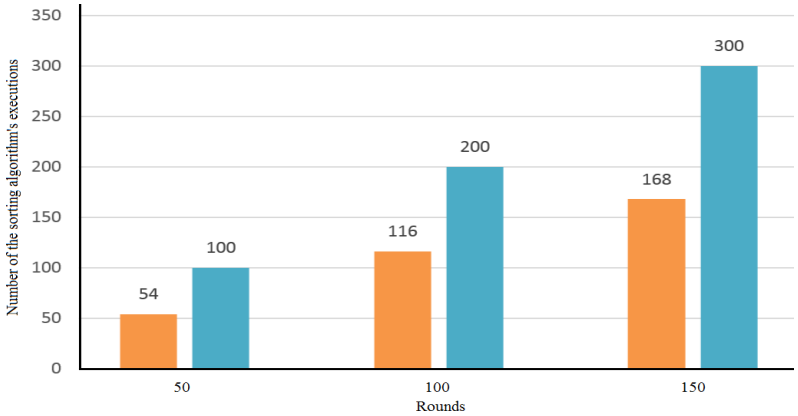


Figure 26: Sorting algorithm execution in the scheduling of Montage workflows with 50 tasks

Figure 27 shows the number of times the sorting algorithm is executed in the basic MFO algorithm and the DMFO-DE algorithm for scheduling the Montage workflows with 50 tasks in 250 rounds. As shown in this figure, our proposed algorithm needs fewer of the Quicksort algorithm's execution. Figure 28 shows the number of times the sorting algorithm is executed in the MFO and the DMFO-DE algorithm to schedule the Montage workflows with 100 tasks. As can be seen in this figure, the DMFO-DE algorithm, which uses the DE algorithm, needs to execute the sorting algorithm fewer times, which makes it much faster than the MFO, and as a result, DMFO-DE can converge more quickly. Figure 29 shows the number of sorting algorithm execution for scheduling the Montage workflows with 1000 tasks. As shown in Figures 26 to 29, since the LA rewards the MFO algorithm for its achieved results, MFO is executed more than DE.

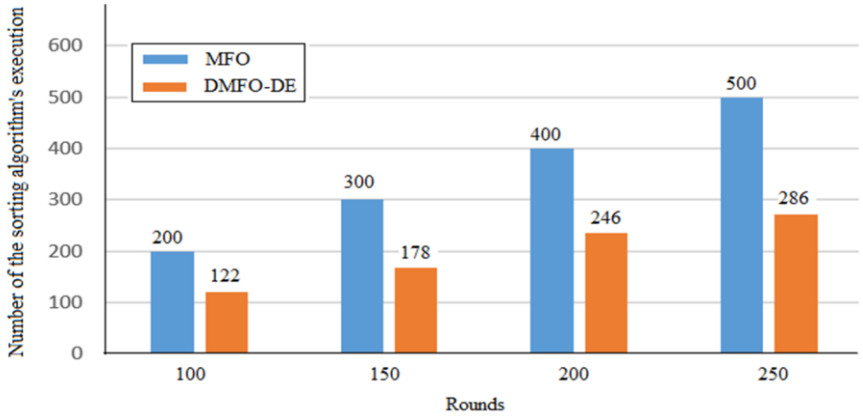


Figure 27: Sorting algorithm execution in the scheduling of Montage workflows with 50 tasks

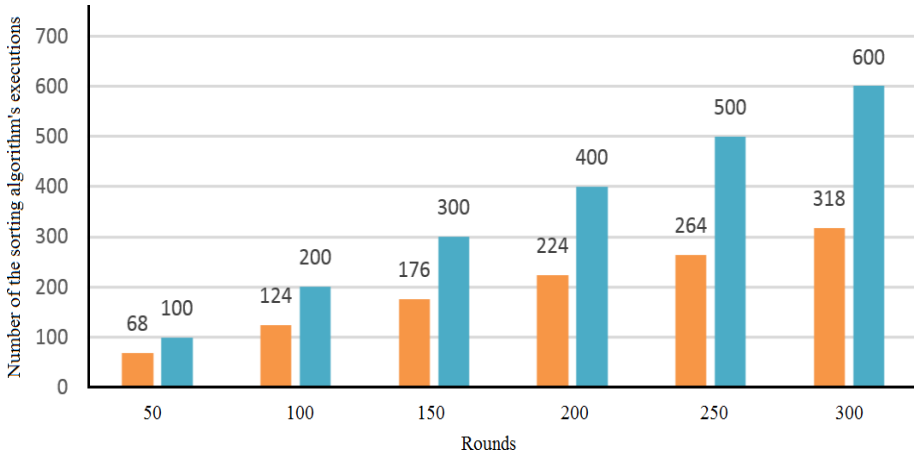


Figure 28: Sorting algorithm execution in the scheduling of Montage workflows with 100 tasks

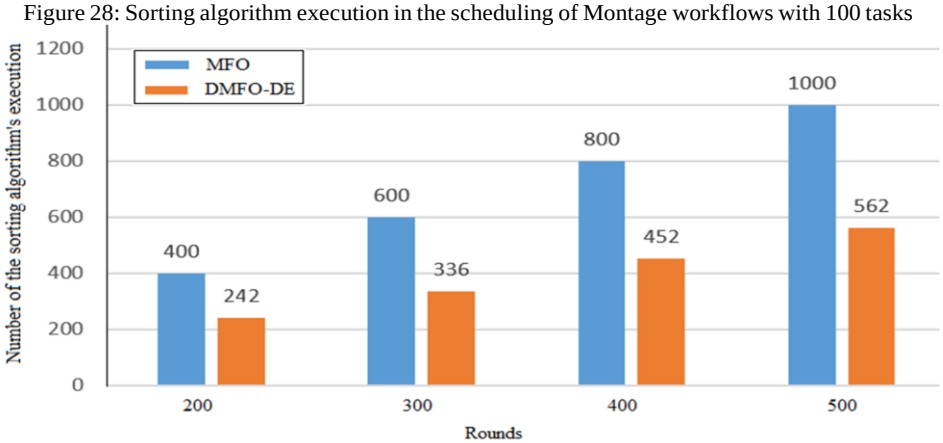


Figure 26: Sorting algorithm execution in the scheduling of Montage workflows with 50 tasks

6. Conclusion

This article presented a discrete and opposition-based version of the MFO algorithm using mutation and crossover operators. Then, we combined it with the DE algorithm and provided a hybrid discrete optimization algorithm called DMFO-DE, which probabilistically execute one of the DMFO and DE algorithms. By using the learning automata, it can decide to run which algorithm more. DMFO-DE incurs less processing overhead than the MFO algorithm and can converge faster. Afterward, we employed the DMFO-DE algorithm for solving the DVFS-based scientific workflow scheduling problem in the fog computing environment. This scheduling approach uses the HEFT algorithm to recognize the priority of tasks in a workflow and then uses the DMFO-DE algorithm to assign the tasks to the most appropriate VMs and allocate the best possible DVFS level VMs. Thus, in addition to reducing the scheduling makespan, we try to minimize workflow scheduling's energy consumption. Extensive simulations are carried out on three types of scientific workflows with different sizes. The obtained results indicated that our solution could outperform workflow scheduling conducted using other optimization algorithms such as MFO, DE, and bat algorithm in terms of the energy consumption and the number of applied VMs. Besides, the results show that our algorithm achieves these results while having less overhead than the basic MFO algorithm.

In future studies, we try to deal with the fog computing environment's workflow scheduling problem by introducing our multi-objective version of our hybrid optimization algorithm. Besides, dealing with multiple fog and environments with unreliable virtual resources is exciting topics that we focus on in the future. Furthermore, enhancing fog computing with multi-cloud environments can be further investigated in future studies.

References

[1] M. Masdari, S. M. Bazarchi, and M. Bidaki, "Analysis of secure LEACH-based clustering protocols in wireless sensor networks," *Journal of Network and Computer Applications,* vol. 36, pp. 1243-1260, 2013.

[2] M. Masdari, S. Barshande, and S. Ozdemir, "CDABC: chaotic discrete artificial bee colony algorithm for multi-level clustering in large-scale WSNs," *The Journal of Supercomputing,* vol. 75, pp. 7174-7208, 2019.

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal,* vol. 1, pp. 22-32, 2014.

[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks,* vol. 54, pp. 2787-2805, 2010.

[5] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering,* vol. 57, pp. 221-224, 2015.

[6] M. Hosseinzadeh, M. Masdari, A. M. Rahmani, M. Mohammadi, A. H. M. Aldalwie, M. K. Majeed*, et al.*, "Improved Butterfly Optimization Algorithm for Data Placement and Scheduling in Edge Computing Environments," *Journal of Grid Computing,* vol. 19, pp. 1-27, 2021.

[7] M. Masdari and A. Khoshnevis, "A survey and classification of the workload forecasting methods in cloud computing," *Cluster Computing,* pp. 1-26, 2019.

[8] M. Masdari and H. Khezri, "Efficient VM migrations using forecasting techniques in cloud computing: a comprehensive review," *Cluster Computing,* pp. 1-30, 2020.

[9] S. Gharehpasha, M. Masdari, and A. Jafarian, "Virtual machine placement in cloud data centers using a hybrid multi-verse optimization algorithm," *Artificial Intelligence Review,* vol. 54, pp. 2221-2257, 2021.

[10] M. Masdari and M. Zangakani, "Green cloud computing using proactive virtual machine placement: challenges and issues," *Journal of Grid Computing,* pp. 1-33, 2019.

[11] S. Gharehpasha, M. Masdari, and A. Jafarian, "Power efficient virtual machine placement in cloud data centers with a discrete and chaotic hybrid optimization algorithm," *Cluster Computing,* vol. 24, pp. 1293-1315, 2021.

[12] A. Shakarami, M. Ghobaei-Arani, A. Shahidinejad, M. Masdari, and H. Shakarami, "Data replication schemes in cloud computing: a survey," *Cluster Computing,* pp. 1-35, 2021.

[13] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13-16.
[14] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review,* vol. 44, pp. 27-32, 2014.
[15] M. R. Hossain, M. Whaiduzzaman, A. Barros, S. R. Tuly, M. J. N. Mahi, S. Roy*, et al.*, "A scheduling-based dynamic fog computing framework for augmenting resource utilization," *Simulation Modelling Practice and Theory,* p. 102336, 2021.
[16] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2014, pp. 325-329.
[17] A. Shahidinejad, F. Farahbakhsh, M. Ghobaei-Arani, M. H. Malik, and T. Anwar, "Context-Aware Multi-User Offloading in Mobile Edge Computing: a Federated Learning-Based Approach," *Journal of Grid Computing,* vol. 19, pp. 1-23, 2021.
[18] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *Journal of Network and Computer Applications,* vol. 98, pp. 27-42, 2017.
[19] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: a hybrid approach," *Cluster Computing,* vol. 24, pp. 319-342, 2021.
[20] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Transactions on Industrial Informatics,* vol. 14, pp. 4712-4721, 2018.
[21] O. H. Ahmed, J. Lu, A. M. Ahmed, A. M. Rahmani, M. Hosseinzadeh, and M. Masdari, "Scheduling of Scientific Workflows in Multi-Fog Environments Using Markov Models and a Hybrid Salp Swarm Algorithm," *IEEE Access,* vol. 8, pp. 189404-189422, 2020.
[22] D. Tychalas and H. Karatza, "A scheduling algorithm for a fog computing system with bag-of-tasks jobs: Simulation and performance evaluation," *Simulation Modelling Practice and Theory,* vol. 98, p. 101982, 2020.
[23] J. C. Guevara and N. L. da Fonseca, "Task scheduling in cloud-fog computing systems," *Peer-to-Peer Networking and Applications,* vol. 14, pp. 962-977, 2021.
[24] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems,* vol. 91, pp. 407-415, 2019.
[25] D. Rahbari and M. Nickray, "Low-latency and energy-efficient scheduling in fog-based IoT applications," *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 27, pp. 1406-1427, 2019.
[26] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, and C. Liu, "Fog computing for energy-aware load balancing and scheduling in smart factory," *IEEE Transactions on Industrial Informatics,* vol. 14, pp. 4548-4556, 2018.
[27] M. S. U. Islam, A. Kumar, and Y.-C. Hu, "Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions," *Journal of Network and Computer Applications,* p. 103008, 2021.
[28] S. Sharma and H. Saini, "A novel four-tier architecture for delay aware scheduling and load balancing in fog environment," *Sustainable Computing: Informatics and Systems,* vol. 24, p. 100355, 2019.

[29] H.-Y. Wu and C.-R. Lee, "Energy efficient scheduling for heterogeneous fog computing architectures," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 2018, pp. 555-560.
[30] W. Wang, G. Wu, Z. Guo, L. Qian, L. Ding, and F. Yang, "Data Scheduling and Resource Optimization for Fog Computing Architecture in Industrial IoT," in *International Conference on Distributed Computing and Internet Technology*, 2019, pp. 141-149.
[31] B. M. Nguyen, H. Thi Thanh Binh, and B. Do Son, "Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment," *Applied Sciences,* vol. 9, p. 1730, 2019.
[32] Y. Wang, C. Guo, and J. Yu, "Immune Scheduling Network Based Method for Task Scheduling in Decentralized Fog Computing," *Wireless Communications and Mobile Computing,* vol. 2018, 2018.
[33] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing,* vol. 4, pp. 26-35, 2017.
[34] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, and M.-T. Zhou, "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks," *IEEE Internet of Things Journal,* vol. 5, pp. 4076-4087, 2018.
[35] D. Hoang and T. D. Dang, "FBRC: Optimization of task scheduling in fog-based region and cloud," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 1109-1114.
[36] D. Rahbari, S. Kabirzadeh, and M. Nickray, "A security aware scheduling in fog computing by hyper heuristic algorithm," in *2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICSPIS)*, 2017, pp. 87-92.
[37] M. Abdel-Basset, D. El-shahat, M. Elhoseny, and H. Song, "Energy-Aware Metaheuristic algorithm for Industrial Internet of Things task scheduling problems in fog computing applications," *IEEE Internet of Things Journal,* 2020.
[38] N. Kaur, A. Kumar, and R. Kumar, "A Novel Task Scheduling Model for Fog Computing," in *Inventive Communication and Computational Technologies*, ed: Springer, 2021, pp. 845-857.
[39] P. Hosseinioun, M. Kheirabadi, S. R. K. Tabbakh, and R. Ghaemi, "A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm," *Journal of Parallel and Distributed Computing,* 2020.
[40] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, and B. M. Nguyen, "An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment," in *Proceedings of the Ninth International Symposium on Information and Communication Technology*, 2018, pp. 397-404.
[41] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling Internet of Things requests to minimize latency in hybrid Fog–Cloud computing," *Future Generation Computer Systems,* 2019.
[42] M. Ghobaei-Arani, A. Souri, F. Safara, and M. Norouzi, "An efficient task scheduling approach using moth-flame optimization algorithm for cyber-physical system applications in fog computing," *Transactions on Emerging Telecommunications Technologies,* vol. 31, p. e3770, 2020.
[43] S. Bitam, S. Zeadally, and A. Mellouk, "Fog computing job scheduling optimization based on bees swarm," *Enterprise Information Systems,* vol. 12, pp. 373-397, 2018.
[44] R. Mahmud and R. Buyya, "Modelling and simulation of fog and edge computing environments using iFogSim toolkit," *Fog and edge computing: Principles and paradigms,* pp. 1-35, 2019.
[45] V. De Maio and D. Kimovski, "Multi-objective scheduling of extreme data scientific workflows in Fog," *Future Generation Computer Systems,* 2020.

[46] R. Ding, X. Li, X. Liu, and J. Xu, "A cost-effective time-constrained multi-workflow scheduling strategy in fog computing," in *International Conference on Service-Oriented Computing*, 2018, pp. 194-207.
[47] S. Ghanavati, J. Abawajy, and D. Izadi, "Automata-based Dynamic Fault Tolerant Task Scheduling Approach in Fog Computing," *IEEE Transactions on Emerging Topics in Computing,* 2020.
[48] M. Masdari and M. Jalali, "A survey and taxonomy of DoS attacks in cloud computing," *Security and Communication Networks,* vol. 9, pp. 3724-3751, 2016.
[49] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, "A survey and classification of the security anomaly detection mechanisms in software defined networks," *Cluster Computing,* vol. 24, pp. 1235-1253, 2021.
[50] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS attacks in cloud computing: Issues, taxonomy, and future directions," *Computer Communications,* vol. 107, pp. 30-48, 2017.
[51] R. Madhura, B. L. Elizabeth, and V. R. Uthariaraj, "An improved list-based task scheduling algorithm for fog computing environment," *Computing,* pp. 1-37, 2021.
[52] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-based systems,* vol. 89, pp. 228-249, 2015.
[53] D. Pelusi, R. Mascella, L. Tallini, J. Nayak, B. Naik, and Y. Deng, "An Improved Moth-Flame Optimization algorithm with hybrid search phase," *Knowledge-Based Systems,* vol. 191, p. 105277, 2020.
[54] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE transactions on Evolutionary Computation,* vol. 13, pp. 398-417, 2008.
[55] S. Gharehpasha and M. Masdari, "A discrete chaotic multi-objective SCA-ALO optimization algorithm for an optimal virtual machine placement in cloud data center," *Journal of Ambient Intelligence and Humanized Computing,* pp. 1-17, 2020.
[56] M. François, T. Grosges, D. Barchiesi, and R. Erra, "Pseudo-random number generator based on mixing of three chaotic maps," *Communications in Nonlinear Science and Numerical Simulation,* vol. 19, pp. 887-895, 2014.
[57] J. A. González and R. Pino, "A random number generator based on unpredictable chaotic functions," *Computer Physics Communications,* vol. 120, pp. 109-114, 1999.
[58] K. S. Narendra and M. A. Thathachar, *Learning automata: an introduction*: Courier corporation, 2012.
[59] Z. Tang, L. Qi, Z. Cheng, K. Li, S. U. Khan, and K. Li, "An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment," *Journal of Grid Computing,* vol. 14, pp. 55-74, 2016.
[60] C.-M. Wu, R.-S. Chang, and H.-Y. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Generation Computer Systems,* vol. 37, pp. 141-147, 2014.
[61] S. Barshandeh, M. Masdari, G. Dhiman, V. Hosseini, and K. K. Singh, "A range-free localization algorithm for IoT networks," *International Journal of Intelligent Systems*.
[62] A. Mohammadzadeh, M. Masdari, F. S. Gharehchopogh, and A. Jafarian, "A hybrid multi-objective metaheuristic optimization algorithm for scientific workflow scheduling," *Cluster Computing,* vol. 24, pp. 1479-1503, 2021.
[63] A. Mohammadzadeh, M. Masdari, and F. S. Gharehchopogh, "Energy and Cost-Aware Workflow Scheduling in Cloud Computing Data Centers Using a Multi-objective Optimization Algorithm," *Journal of Network and Systems Management,* vol. 29, pp. 1-34, 2021.
[64] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in engineering software,* vol. 95, pp. 51-67, 2016.

[65] K. Asghari, M. Masdari, F. S. Gharehchopogh, and R. Saneifard, "A chaotic and hybrid gray wolf-whale algorithm for solving continuous optimization problems," *Progress in Artificial Intelligence,* pp. 1-26, 2021.