Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Job scheduling problem in fog-cloud-based environment using reinforced social spider optimization

P. Kuppusamy[1], N. Marline Joys Kumari[2], Wael Y. Alghamdi[3], Hashem Alyami[3], Rajakumar Ramalingam[4], Abdul Rehman Javed[5,6] and Mamoon Rashid[7*]

Abstract

Fog computing is an emerging research domain to provide computational services such as data transmission, application processing and storage mechanism. Fog computing consists of a set of fog server machines used to communicate with the mobile user in the edge network. Fog is introduced in cloud computing to meet data and communication needs for Internet of Things (IoT) devices. However, the vital challenges in this system are job scheduling, which is solved by examining the makespan, minimizing energy depletion and proper resource allocation. In this paper, we introduced a reinforced strategy Dynamic Opposition Learning based Social Spider Optimization (DOLSSO) Algorithm to enhance individual superiority and schedule workflow in Fog computing. The extensive experiments were conducted using the FogSim simulator to generate the dataset and an energy-efficient open-source tool utilized to model and simulate resource management in fog computing. The performance of the formulated model is ratified using two test cases. The proposed algorithm attained the optimized schedule with minimized cost function concerning the CPU processing period and assigned memory. Our simulation outcomes show the efficacy of the introduced technique in handling job scheduling issues, and the results are contrasted with five existing metaheuristic techniques. The results show that the proposed method achieves 10% - 15% better CPU utilization and 5%-10% less energy consumption than the other techniques.

Keywords: Social spider algorithm, Job scheduling, Fog computing, Cloud computing

Introduction

The everyday needs of society are facilitated using cloud computing [1]. Cloud computing system uses master-slave infrastructure among service providers. The client users access cloud computing on-demand to access the service using "pay-as-you-go" [2]. Cloud services are assessed anywhere at any time through the internet. Besides their benefit, cloud computing consumes high latency and higher network bandwidth, since data centers are available far from the client user. A novel architecture is to integrate fog computing in the cloud environment to eradicate the above-said issue. The idea of fog computing extends direct communication among edge nodes of the network [3]. Fog computing network consists of fog servers with a wireless communication device, data storage memory and computational unit [4]. The energy-efficient resource utilization plays a significant role in this network. Therefore, any deviation in cloud data might result in a substantial change in energy consumption, affecting starvation. This situation can be eradicated by efficiently scheduling the jobs into the fog server. But it is observed that dynamic job scheduling is tricky due to complicated task flow and perplexing issues [5].

Job scheduling problem (JSP) is an NP-hard combinatorial optimization problem due to various numbers of

*Correspondence: mamoon873@gmail.com

[7] Department of Computer Engineering, Faculty of Science and Technology, Vishwakarma University, Pune 411048, India
Full list of author information is available at the end of the article

all feasible solutions is $t!^{pm}$ where t denotes the count of jobs/tasks, and pm specifies the total count of physical machines. Individual jobs deal with the unique operation sequence. The main aim of job scheduling is to reduce the response time and maximise the company's revenue. However, the JSP suffers two main challenges: choosing an adequate facility and determining the best order of operations to reduce the makespan. Several researchers have contributed to solving job scheduling using mathematical models, namely exact, heuristic and meta-heuristic approaches. Firstly, Exact strategies like the dynamic approach, branch and bound method, and component and price approaches are adequate to solve the small-scale problem and to provide optimal solutions. However, these exact approaches are time-consuming and suffer to solve large-scale real-time issues. The minor JSPs problems can be optimally solved using the CPLEX application [6].

In the real-time, most JSPs are large scale and generic mathematical models cannot guarantee optimal solutions. Still, they can provide near-optimal solutions concerning colossal time consumption. Recent research has used meta-heuristic algorithms, namely evolutionary algorithm and swarm intelligence algorithm, to solve the JSPs problem. A hybrid Genetic algorithm [7] was utilized to minimize the maximum completion time. However, the model fails to provide the optimal solution due to poor convergence. The multi-objective artificial bee colony algorithm [8] was introduced to address the multi-objectives, assignment vector, job permutation and machine speed selection. The Pareto archive particle swarm optimization algorithm [9] combines the Crowing distance-based archive and mutation process to handle the JSP issues. Due to the method's limited exploration ability, it becomes stuck in local optima. In [10], the ant colony method was used to overcome the problems with flexible job shops. The outcomes demonstrate that the strategy produces superior results to the compared algorithms.

The hybrid genetic and cuckoo search technique devised by Singh Satyendra et al. [11] produces better results in small-scale instances but falls short in medium- and large-scale ones. Wang et al. [12] used hybrid discrete differential evolution to handle task scheduling challenges and reduce the maximum turnover time. However, the algorithm is significantly more complicated because of the problem with the solution update. Lu et al. [13] suggested a cellular grey wolf optimisation to reduce cycle time in blocking flow shops. To strengthen the search process by adding new control factors, the author of [14] recently presented hybrid whale optimization with the Levy flight and DE algorithm. They also used the technique to shorten the JSP problem's turnaround time. The spotted hyena optimizer was devised to address the discrete work sequence difficulties [5]. Atay et al. [15] suggested a clonal selection algorithm to address the JSP difficulties. The proposed approach yields superior results in terms of reducing completion time.

The main objective of the flower pollination algorithm [16] is to shorten the time complexity of task sequence issues. The author in [17] introduced the bi-objective optimization technique to lower the makespan and cost factors. The author also used the goal programming method to submit multi-objective task sequence assignments in cloud-fog computing [18]. The simulation results offer superior outcomes in terms of access level control, deadline measures, and service delay times. A modified pigeon-inspired optimization technique [19] was applied to deal with job sequence issues. The results of the experiment outperform the compared algorithms. However, due to the complexity of the algorithms, task assignment takes a lengthy time.

Social Spider Optimization (SSO) technique is a recently introduced swarm intelligence algorithm that mimics the social conduct of a spider. The technique deliberates the working principle of spiders concerning the biotic actions of cooperative groups. The SSO approach segregates the total population into numerous search agents, namely male and female spiders and adequately processes the specialized operators for each search agent. These operators aid the algorithm in trade-off the search process towards finding the optimal solutions. Initially, SSO was introduced to handle global optimization problems [20]. In later work, some researchers modified the generic SSO algorithm to solve binary optimization problems, namely clustering and classifications [21]. Further, the generic SSO algorithm has been utilized for several complex problems, namely constraint optimization problems [22], load balancing in a grid environment [23], clustering analysis [24], and multi-level thresholding problems [25].

The major highlights of this work are stated as follows;

- Our proposed technique integrates the dynamic opposition-based learning technique into the social spider algorithm to enhance the exploration capability and to maximize the convergence rate.
- This novel DOLSSO technique handles job scheduling issues in cloud environments. It enhances the diversity of scheduling solutions and reduces the local optimal struck.
- Extensive experimentation was conducted using FogSim with two different test scenarios to validate the efficacy of the proposed system in handling the job scheduling problem.

- The solution quality of the proposed algorithm is compared with five recent state-of-art meta-heuristic algorithms.
- The results show that the proposed method achieves 10% - 15% better CPU utilization and 5%-10% less energy consumption than the other techniques.

The rest of the article is structured as Literature work section deliberates existing works in the literature on fog computing infrastructure for handling scheduling problems. The JSP in FOG computing section discusses the system prototype and Proposed methodology section presents the problem definition and shows a representation of the SSO algorithm to solve job scheduling problems, and Experimental analysis section illustrates the simulation analysis. Finally, Conclusion section concludes the work with its outcome and future directions.

Literature work

The scheduling issue is the sharing of present resources concerning the count of tasks/jobs. The JSP is a significant problem and complex, like the scheduling problem. The job scheduling problem on a heterogeneous system is considered an NP complex problem [26]. Graph partitioning strategy is adapted to balance the load in fog computing infrastructure [27]. The authors used dynamic graph partitioning for load balancing and cloud automation technology for building virtual machines. Data stream processing [28] is implemented in fog computing with QoS aware scheduler, and it will monitor incoming and outgoing data for node utilization and inter-node communication. Load balancing in fog computing creates an issue with the quality of experience [29]. The authors proposed a reduced task scheduling algorithm to process load balancing in fog computing. Based on the scheduling process, resources are allocated to serve fog nodes called small cells. Van den Boss et al. [30] focus on the cost-efficiently schedule based on deadline constraint heuristic on public and private clouds. The budget constraint schedule is implemented in the cloud environment and is based on the advantage of comparative methodology. This algorithm suits extensive scale networks due to its higher complexity [31].

In a standard JSP problem, there is the job block which determines the tasks and a block of physical systems with its resources [32]. In JSP, the general strategy is to process each job in the sequence of physical machines with the allocated resource constraints. However, allotted resource constraints are not sufficient in real-world situations. Economic factors and globalization force the IT industries to sustain by modifying single-standard facilities into multi-facility production. Hence, several researchers have started to perform their research on JSP, which is a much more perplexing problem and schedule jobs by minimizing production cost and computational time [33].

Based on recent research works, it is examined that several researchers have worked on single objective constraints such as reducing makespan. Further, in multi-objective restrictions such as reducing makespan, increasing workload, and overall power utilization of companies). Hongtao Tang et al. [34] introduced hybrid teaching and learning-based approach to solving the scattered and casting job scheduling issue. The author presented a three-layer coding method and five neighbourhood formations to handle the local search process in the algorithm. A real-time casting enterprise case study evaluates the algorithm's performance. The observed results are compared with the six state-of-art metaheuristic algorithms to prove their efficacy. Hui Li et al. [35] proposed a hybrid differential evolution algorithm to tackle adaptable work planning by reevaluating tasks and job priority limitations. Chromosome encryption and decryption strategy are introduced to handle the real-time constraints and to attain a high-quality initial population. Further, genetic operators with self-adjusting conditions are incorporated to widen the search regions and increase the convergence rate.

SSO algorithm is a recently proposed metaheuristic algorithm that works based on the working principle of the social spider. Erik et al. [22] proposed an SSO algorithm to solve global continuous optimization problems. A robust clustering approach using SSO with a chaotic technique was presented in 2018 [36]. The literature review discussed above deliberates that few works are carried out to solve JSPs in fog computing using metaheuristic techniques. In addition, none of the researchers has utilized the SSO algorithm in literature to handle the JSP issue. Hence, we introduced a novel discrete SSO to run JSP, and the outcome specifies that DOLSSO has attained a better result than existing methods.

The JSP in FOG computing

In this section, the overview of JSP is discussed with the set of jobs, sequences and challenges in JSP. Later, the system design for fog computing infrastructure is illustrated. In addition, the mathematical formulation of large-scale JSP is represented.

The overview of JSP

JSP is considered a significant scheduling issue that needs to process the sequence of jobs on the system or server by considering the number of constraints and objectives. A $\setminus \times \ell$ JSP can be expressed as a set of $\setminus$ jobs $\psi = \{\psi_1, \psi_2, \ldots, \psi_\setminus\}$ that is executed on a set of fog servers $\Upsilon = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_\ell\}$. Each task ψ_i is expressed by

order of $\backslash_i$ process $P_{ij}\left(j = 1, 2, \ldots, \backslash_i\right)$ to be processed in a sequence completion order of jobs in a scheduled list. The challenge of JSP is to identify the adequate server for each process (known as Server selection) and the process execution order on the server (known as process ordering). According to the constraints of server election and job execution on one or more servers. The JSP is classified into three major classifications: JSP, partial-JSP and total-JSP. In JSP, the jobs are executed only on one elected server, whereas partial-JSP jobs can be performed on one or more partial machines. Finally, total-JSP is hugely flexible and can be implemented on any number of servers, as shown in Fig. 1.

System design

Fog computing infrastructure consists of \ number of the fog server machine. In our architecture, the cloud administrator is responsible for partitioning ψ job into T tasks. The decomposition of employment into the scheme is based on the number of available Fog server machines. We propose an SSO algorithm executed by the cloud administrator to ensure optimal scheduling in the Fog server.

Job scheduling is represented by a Directed Weight Graph (DWG); DWG is a weighted-node directed graph denoted by $DWG = (T, E, F, \mu, \delta)$. T represents a set of tasks to be performed, and E indicates data dependencies among tasks. Let cloud consist of Υ set of Fog server, $\Upsilon = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_\ell\}$. There are \ jobs submitted to schedule and executed by the Fog server. The jobs are portioned into tasks, and each assignment is apportioned to one fog server to schedule. Let Job ψ be partitioned into tasks and represented as $\psi = \left\{\psi_1, \psi_2, \ldots, \psi_\backslash\right\}$, where $\backslash = |\ \mathrm{Y}\ |$ .E represents the set of edges among nodes in T.

Definition 1

In DWG, assume any task can be processed on any Fog server machine in the formulated architecture. The numeral of edges in the graph is denoted by $|E| \subseteq |\Upsilon| \times |T|$. Each task T_i is processed on the Fog server machine concerning task execution time and allocated memory to process, and it is represented as, $T_i = \begin{bmatrix} \mu_i \\ \delta_i \end{bmatrix}$. The CPU execution time of the task T_i on Fog, server is represented as μ_i. δ_i is used to represent allocated memory for the task T_i to perform.

Definition 2

In DWG, input for the graph is assigned using the Initialization Matrix (I_G) $I_G = (\Upsilon_{i,j})_{n*n}$. The initialization matrix consists of i task to be executed on j fog server machine. The element in the initialization matrix is one if the job is assigned to the fog server or else it is 0, and it can be represented as $\Upsilon_{i,j} \in 0, 1$.

$$\Upsilon_{i,j} = \begin{cases} 1, & \textit{if task is allocated to Fog server machine} \\ 0, & \textit{otherwise} \end{cases} \tag{1}$$

The objective of job scheduling is minimizing the total cost of execution time on processing tasks and, thus Optimal Scheduled Matrix (O_G) is given as $O_G = \left(\Upsilon_{ij}\right)_{n*n}$.

The mathematical formulation of large-scale JSP

This section discusses the mathematical formulation of the large-scale job scheduling problem. Large-scale JSP is an extension of JSP in the number of jobs and machines. Therefore, Large scale JSP consists of the same mathematical model of JSPs. The constraints and objective functions of large-scale JSP are mathematically modelled and given as follows.

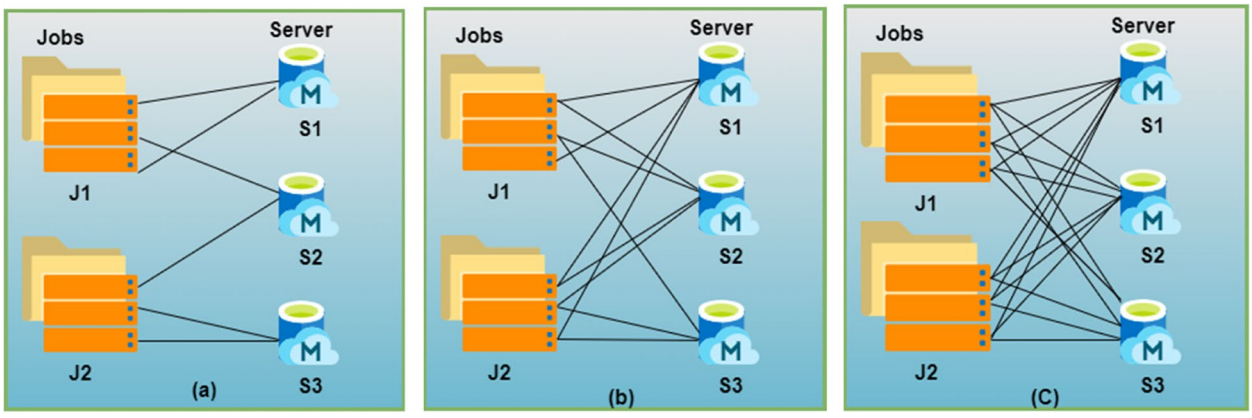

Fig. 1 **a** JSP (**b**) P-JSP (**c**) T-JSP

1) Once the i^{th} job of j^{th} the task is assigned to the ℓ^{th} server, it cannot be interrupted until it completes its execution. Then, the P_{ij} is computed as an additive of its initial time and handling time.

$$C_{ij\ell} = S_{ij\ell} + E_{ij\ell} \tag{2}$$

where, $C_{ij\ell}$, $S_{ij\ell}$ and $E_{ij\ell}$ are denoted as culmination time, initial time and handling time of the j^{th} task of the i^{th} job P_{ij} on server ℓ, respectively.

2) The initial time of the i^{th} job of j^{th} task P_{ij} on the server, ℓ is calculated based on the peak period of the earlier task μ on the server and the peak time of the earlier task.

$$S_{ij\ell} = \max\left\{S_{(i-1)\mu\ell} + E_{(i-1)\mu\ell}, C_{i(j-1)(\ell-1)}\right\} \tag{3}$$

3) Server ℓ closes down when the last jobs are completed its execution.

$$\Psi_\ell = \Psi_{\vartheta\ell} \tag{4}$$

where, Ψ_ℓ denotes the shutdown time of the server ℓ, ϑ indicates the end job on sever ℓ, $\Psi_{\vartheta\ell}$ represents the peak time of the previous job on server ℓ.

4) Some set of limitations on the server for the execution of the process. Thus, the constraints are specified below.

$$\begin{aligned} &C_{\omega\ell} - C_{i\ell} + \mathrm{Y}(1 - \chi_{\mathrm{i}\omega\ell}) \geq \mathrm{E}_{\omega\ell} \\ &i, \omega = 1, 2, \ldots, n;\ \ell = 1, 2, \ldots, m \end{aligned} \tag{5}$$

where n and m indicate the sequence of jobs and servers, respectively. $C_{\omega\ell}$, $E_{\omega\ell}$ specifies the culmination time and handling time of job ω on server ℓ, Υ determines the sizeable positive integer. $\chi_{i\omega\ell}$ denotes the handling sequence priority of job i and job ω on server ℓ. If job i Is executed on the server ℓ then $\chi_{i\omega\ell} \in 1$, otherwise 0.

5) A single server suffers from handling more than one operation simultaneously.

$$S_{ij\ell} \geq C_{\omega t\ell} - \mathrm{T}_{ij\omega t\ell}.\mathrm{Y}, \forall\ell \in \mathrm{Y}_{\mathrm{ij}} \cap \mathrm{Y}_{\omega\mathrm{t}} \tag{6}$$

$$S_{\omega t\ell} \geq C_{ij\ell} - \left(1 - \mathrm{T}_{ij\omega t\ell}\right).\mathrm{Y}, \forall\ell \in \mathrm{Y}_{\mathrm{ij}} \cap \mathrm{Y}_{\omega\mathrm{t}} \tag{7}$$

where, $\mathrm{T}_{ij\omega t\ell}$ denoted as whether the i^{th} job of the j^{th} task P_{ij} is handled earlier than the t^{th} task of the ω^{th} job $P_{\omega t}$ on server ℓ, if it is executed, then 1; otherwise, 0. $\Upsilon_{ij} \cap \Upsilon_{\omega t}$ indicates the vacant server set that can able to handle the tasks P_{ij} and $P_{\omega t}$ simultaneously.

6) When the task has numerous operations on a similar job, it is essential to process through the determined order.

$$\begin{aligned} &S_{ij\ell} \geq C_{i(j-1)\ell}, \ell \in \Upsilon_{ij} \\ &\forall_i = 1, 2, \ldots, n, \forall_j = 2, \ldots, \varphi_i \end{aligned} \tag{8}$$

where, ϕ_i denotes the numeral tasks of job i.

7) When the server is scheduled, then the entire jobs can access at the initial time ($S = 0$)

$$S_{ij\ell} \geq 0 \tag{9}$$

$$C_{ij\ell} \geq 0 \tag{10}$$

8) For each job, one server can be chosen for its execution.

$$\sum_{\ell=1}^{m} \beta_{ij\ell} = 1 \tag{11}$$

where $\beta_{ij\ell}$ represented as whether the server ℓ handles the job P_{ij}, one deliberates the process is executing, 0 determines the process is not completed.

The objective of the JSP optimization problem incorporates max culmination time, max server load, min execution cost, etc. This work utilizes the max culmination time to ensure the proposed algorithm's efficacy, which can be mathematically modelled as follows.

$$F = \min(Makespan) = \min_{1\leq\ell\leq m}\max\left\{\max_{1\leq i\leq n}\Psi_\ell\right\} \tag{12}$$

where Makespan denotes the max culmination time of all servers, it is the initial specification to calculate the handling efficacy. Ψ_ℓ is mathematically expressed as.

$$\Psi_\ell = \sum_{i=1}^{n}\sum_{j=1}^{\varphi_i}\sum_{\ell=1}^{m}\left(S_{ij\ell}\beta_{ij\ell} + E_{ij\ell}\beta_{ij\ell}\right) \tag{13}$$

Proposed methodology

This section discusses the overview of generic social spider optimization algorithm, the dynamic opposite learning concept and the formulation of the proposed algorithm.

Overview of social spider optimization algorithm

SSO algorithm mimics the foraging actions of vibration sense. The source of food is another social spider or prey on the web. The direction of the food source is erudite using vibration sense. The spider denotes a feasible solution in the SSO algorithm, and the web indicates search space. SSO consists of male and female agents; the female numeral individual is more significant than the male individual. The numeral female individual (N_f) and male individuals (N_m) are calculated using

$$N_f = Floor[(0.9 - \text{rand}.0.25).Np] \tag{14}$$

$$N_m = Np - N_f \tag{15}$$

where *rand* denotes the arbitrary function within the range of [0,1]; *Np* denotes the numeral populations; each spider receives weight W_i in population, Φ using fitness calculation. The weight was calculated using

$$W_i = \frac{fit(\Phi) - worst_\Phi}{best_\Phi - worst_\Phi} \tag{16}$$

The vibration sense $V_{a,b}$ on the web received by spider *a* and sent by spider b. $d_{a,b}$ is the Euclidean distance, which can be calculated using

$$V_{a,b} = W_i.e^{-d_{a,b}^2} \tag{17}$$

The cooperative behaviour of female agents is calculated using

$$f_i(t+1) = \begin{cases} f_i(t) + x.V_{i,c}.(\Phi_c - f_i(t)) + y.V_{i,d}.(\Phi_d - f_i(t)) \\ \quad + z.\left(\text{rand} - \frac{1}{2}\right) r_l < \gamma \\ f_i(t) - x.V_{i,c}.(\Phi_c - f_i(t)) - y.V_{i,d}.(\Phi_d - f_i(t)) \\ \quad + z.\left(\text{rand} - \frac{1}{2}\right) r_l \geq \gamma \end{cases} \tag{18}$$

where *x*, *y*, *z*, r_l, *rand* are arbitrary values between [0,1], γ is the threshold value, *t* is the iteration number for nearest spider *c*, *d*. The cooperative behaviour of male agents is calculated using

$$m_i(o+1) = \begin{cases} m_i(o) + x.V_{i,f}.(\Phi_f - m_i(o)) + +z.\left(\text{rand} - \frac{1}{2}\right) \\ \quad if\ W_{N_{f+i}} > W_{N_{f+m}} \\ m_i(o) + x.\left(\frac{\sum_{e=1}^{N_m} m_e(o).W_{N_{f+h}}}{\sum_{e=1}^{N_m}.W_{N_{f+h}}} - m_i(o)\right) \\ \quad if\ W_{N_{f+i}} \leq W_{N_{f+m}} \end{cases} \tag{19}$$

where, $\frac{\sum_{e=1}^{Nm} m_e(o).W_{N_{f+h}}}{\sum_{e=1}^{Nm}.W_{N_{f+h}}}$ denotes the weighted mean for the male population.

Parameter setting

Initialize the population Φ, threshold value of γ, and maximum iteration.

Calculate the number of female and male spider agents using Eq. (14, 15).

$Iter = 1$

Population Initialization

Initialize randomly male and female spiders. Calculate the mating radius using Eq. (12).

Solution Evaluation

Calculate the weight of the spider agent using Eq. (16)

Female Operator

Vibration calculation for a local and global best solution using Eq. (18)

Male operator

The cooperative behaviour of male spiders is calculated using Eq. (19)

Mating Operation

Calculate the radius of mating using Eq. (20)

Using the Roulette method regenerates a new population.

Until $Iter$ = Maximum number of Iteration

The dominant male and female agents are used for mating within radius r can be calculated using

$$r = \frac{\sum_{j=1}^{n}\left(ub_j^{high} - lb_j^{low}\right)}{2.n} \tag{20}$$

where ub_j^{high} and lb_j^{low} are the upper and lower confines of the problem. The SSO algorithm is presented in algorithm 1.

Dynamic opposite learning method

The dynamic opposite learning method (DOL) is an extension of the opposition-based learning (OBL) method that is proposed by Xu et al. [37]. However, the OBL method can improve the quality of the algorithm towards optimal concerning the exploration of search space. But, determining the opposite position from the current situation is imminent. DOL is familiarized to advance the solution quality by eliminating the local optimal struck to eradicate the issue. The DOL is moreover similar to OBL by changing the opposite number X_o with X_{ro} (i.e., $X_{ro} = rand\,.\,X_o, rand \in [0,1]$). The stable pursuit region can be modified into an unstable pursuit region trademarking the active change concerning arbitrary number.

Assumption

Let α denoted as the lower bound of the individual, λ denotes the upper bound of the individual. For instance, X_{ro}, α and λ consist of two cases, namely $X_{ro} \in [\alpha,\lambda]$ and $X_{ro} \notin [\alpha,\lambda]$. The individual X_{ro}, X_o and X consist of three associations such as 1) X_{ro} between X_o and X 2) X_{ro} is larger than X_o and 3) X_{ro} is minimum than X.

The dynamic opposite number X_{do} can be regenerated when $X_{do} \notin [\alpha,\lambda]$. At the same time, if X_{ro} is exceed the upper and lower bound due to the below Eq. (21).

$$X_{do} = \left\{ \begin{array}{l} X + r_1.(X_{ro} - X) \;\; if \; X_{ro} > \lambda \\ X + r_2.(X_{ro} - X) \;\; if \; X_{ro} < \alpha \end{array} \right\} \tag{21}$$

The mathematical model of DOL concerning the dynamic opposite number and the active opposite point is expressed as follows.

Dynamic opposite value

It is considered to determine the step-by-step process of DOL. The mathematical model of X_{do} can be expressed in Eq. (22).

$$X_{do} = X + z.r_1.(r_2.X_o - X) \tag{22}$$

$$X_o = \lambda + \alpha - X \tag{23}$$

where, r_1 and r_2 denoted as arbitrary values with the bound of [0,1]; z indicates the learning weight value; X represented as the actual number with the range of $[\alpha,\lambda]$.

Dynamic opposite point: The expansion of positive contradictory value in the various dimensional search point. The mathematical model of the active opposite end can be expressed in Eq. (24).

$$X_{do,j} = X_j + z.r_3.\left(r_4.X_{o,j} - X_j\right), j = 1, 2, \ldots, D \tag{24}$$

$$X_{o,j} = \lambda_j + \alpha_j - X_j \tag{25}$$

where, D determines the D-dimensional space that incorporates various possible individuals; X specifies the present individual with D-dimensional space (i.e., $X = \{X_1, X_2, \ldots, X_D\}$ are limited to upper (i.e., $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_D\}$ and lower $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_D\}$ bound; r_3 and r_4 are indicates the arbitrary values with the range of [0,1].

The DOL method is incorporated into the beginning and rehearsal of the SSO population. In the initialization process, the opposite population Φ^{do} is generated by the population Φ (i.e., $\Phi^{do} \cup \Phi$). Then, the fitness of the population $\Phi^{do} \cup \Phi$ is computed and picks the half set of individuals to the population Φ^S. In the generation process, if the hopping process gratifies, it produces the Φ^{do} by Φ^S and process the whole population as $\Phi^{do} \cup \Phi^S$, then compute the fitness of $\Phi^{do} \cup \Phi^S$ and repeat the process until the process completes. If the solution of Φ^{do}, Φ^S and Φ exceeds the boundary, then regenerate the individual within the upper and lower limit.

DOL-based SSO algorithm

In this part, we proposed a novel variant of SSO, namely DOLSSO, to enhance the standard SSO algorithm to handle the precocity that constantly riddles several optimization techniques.

Population initialization based on DOL

The opposite solutions based on DOL are generated from the set of the first half of present individuals. The DOL population is processed with the SSO population initialization method. The mathematical model is expressed as follows.

$$\Phi_{ij}^{do} = \Phi_{ij} + r_{1,i}.\left(r_{2,i}.\left(\lambda_j + \alpha_j - \Phi_{ij}\right) - \Phi_{ij}\right) \tag{26}$$

where, Φ_{ij} indicates the j^{th} dimension of i^{th} the solution in population created by SSO, $r_{1,i}$ and $r_{2,i}$ are two arbitrary values within the range of [0,1], λ_j and α_j specifies the upper and lower boundary of j^{th} dimension of i^{th} solution

Φ_{ij}^{do}, determines the opposite individual of Φ_{ij}. The DOL population initialization Φ_{ij}^{do} should satisfy the boundary region as per Eq. (27).

$$\Phi_{ij}^{do} = RN, if\ \Phi_{ij}^{do} < \alpha_j \Big\| \Phi_{ij}^{do} > \lambda_j \tag{27}$$

where RN indicates the arbitrary values within the limit of $[\alpha_j, \lambda_j]$.

Generation hopping based on DOL

DOL iteration hopping is analogous to the DOL population initialization process. This DOL iteration hopping strategy

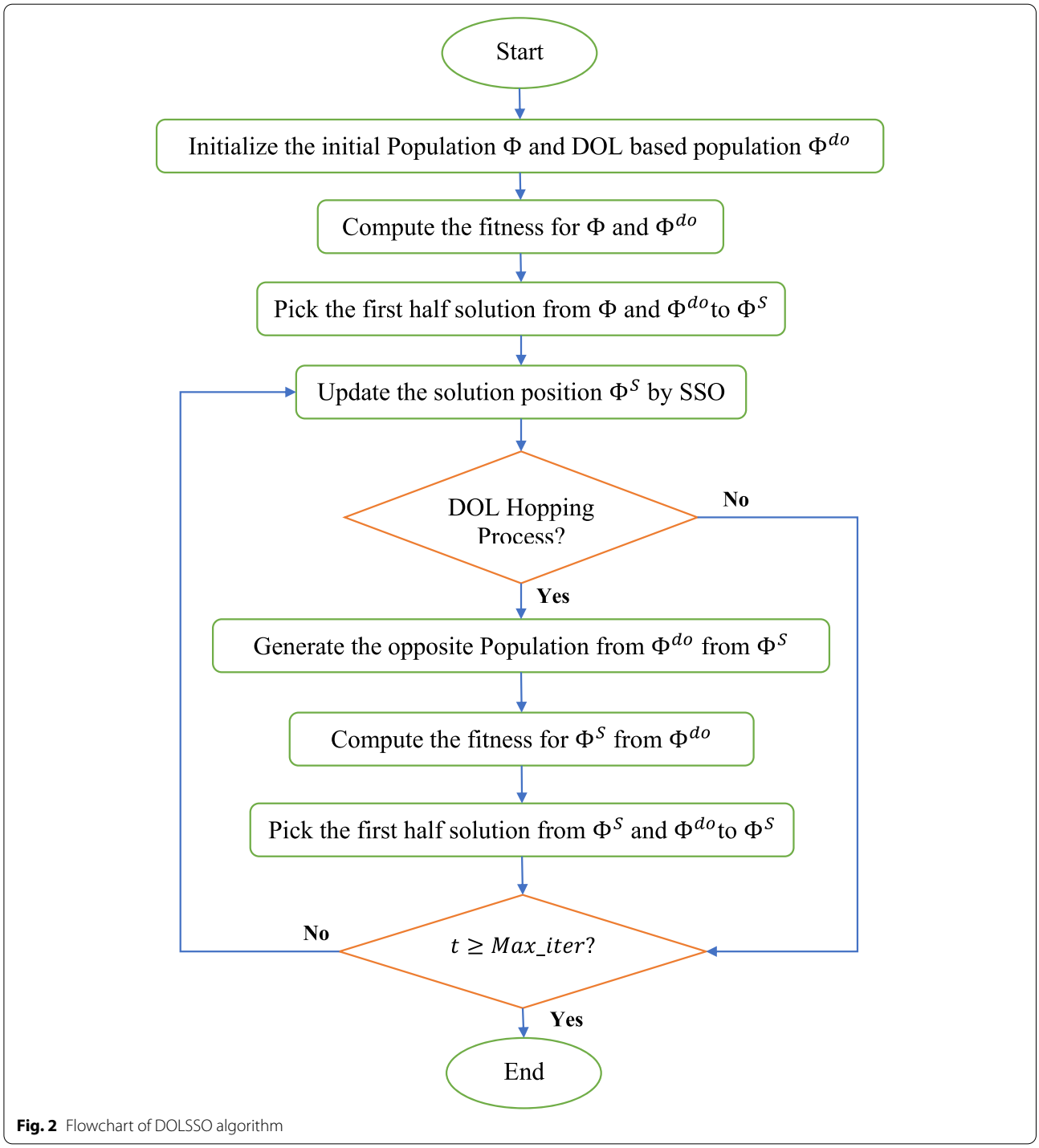

Fig. 2 Flowchart of DOLSSO algorithm

can be processed into whole generation to aid the algorithm to eradicate from local optimal struck. The hopping rate of the DOL method is represented by $\delta \in [0,1]$ to determine the probability of the technique according to the hop rate. If the arbitrary value produced is minimal than the hopping rate factor δ, DOL process the hopping action. The mathematical model of the hopping process is expressed as follows.

$$\Phi_{ij}^{do} = \Phi_{ij} + z.\ r_{3,i}.\left(r_{4,i}.\left(\lambda_j + \alpha_j - \Phi_{ij}\right) - \Phi_{ij}\right) \quad (28)$$

where z denotes the learning weight that differs concerning various scenarios and conditions, $r_{3,i}$ and $r_{4,i}$ are the two arbitrary values within the range of [0,1].

1: Generate arbitrary initial Population Φ;
2: ***For*** $i = 1: Np$
3: $r_{1,i} = rand(0,1), r_{2,i} = rand(0,1)$;
4: ***For*** $j = 1: D$
5: $\Phi_{ij}^{do} = \Phi_{ij} + r_{1,i}.(r_{2,i}.\left(\lambda_j + \alpha_j - \Phi_{ij}\right) - \Phi_{ij})$;
6: Ensure the boundary limits;
7: ***End For***
8: ***End For***
9: Pick the top best N solutions from $\Phi^{do} \cup \Phi$ to Φ^{S}
10: $t = 1$; # Initial Iteration t is fixed as 1
11: ***while*** $t \leq Max\_Iter$
12: Compute the fitness of all solutions;
13: ***For*** $i = 1: Np$
14: Compute N_f, N_m, W_i, $V_{a,b}$ using Eq. (14) to Eq. (17);
15: Update the cooperative behaviour of female agents using Eq. (18);
16: Update the cooperative behaviour of male agents using Eq. (19);
17: Determine r using Eq. (20);
18: Ensure the boundary limits of all solutions;
19: ***End for***
20: ***For*** $i = 1: Np$
21: ***If*** $rand < \delta$
22: $r_{3,i} = rand(0,1), r_{4,i} = rand(0,1)$;
23: ***For*** $j = 1: D$
24: $\Phi_{ij}^{do} = \Phi_{ij}^{S} + z.\ r_{3,i}.(r_{4,i}.\left(\lambda_j + \alpha_j - \Phi_{ij}\right) - \Phi_{ij}^{S})$;
25: Ensure the boundary limits;
26: ***End for***
27: ***End If***
28: ***End for***
29: Pick the top best N solutions from $\Phi^{do} \cup \Phi^{S}$ to Φ^{S};
30: ***End while***

Table 1 Simulation parameters for experimentation

Parameters	Values
Number of processors	4, 8, 12, 16, 20, 24
Number of Tasks	20, 40, 80, 160, 320, 400
Number of fog nodes	5, 10, 15, 20
DOLSSO (Proposed)	Max_iter: 100, Population size: 60, Hopping rate factor (δ): 0.5, Learning weight (z): 0.1, Upper limit (λ): 1
SSO [22]	Learning weight (z): 0.1 Coefficient parameter: [0,2]
GOA [39]	Attraction force: [2.079,4], Repulsion factor: [0, 2.079], coefficient value: [1, 0.0001]
SSA [40]	Step size: 10, Fitness function constant: 0.9
GWO [41]	coefficient parameter (a): [2,0]
WOA [42]	Parameter (A): [−1, 1] Random probability (p): 0.5

DOLSSO algorithm

As discussed earlier, DOLSSO is the modified version of SSO that includes the DOL method in standard SSO. The SSO algorithm is given in algorithm 1, and the DOL method is discussed in Dynamic opposite learning method section. The procedure of DOLSSO is presented in algorithm 2, and the workflow of DOLSSO is sketched in Fig. 2.

Decoding and encoding for JSP

A wide variety of strategies are available to encode and decode the JSP. Gao et al. [38] introduced one of the most popular strategies, including server selection and process ordering vectors. Although the strategy provides better results for some scenarios, its representation of individuals increases memory utilisation. In this work, we utilized the encoding and decoding strategy from the reference [37].

Exploration and exploitation analysis

We utilized the hopping rate factor (δ) in the proposed system to trade off the exploration and exploitation process. The individual that satisfies the hopping element then the current individual will undergo the exploration process using the DOL method. Otherwise, the individual utilizes the SSO method to exploit the search space. Therefore, we use δ with the fixed values of 0.5 to determine the solution update process. The DOLSSO algorithm initiates with a set of random solutions. At each generation, search individuals update their positions concerning randomly selected search agents, or the best individual found so far. Depending on the hopping factor δ, the proposed algorithm can switch between the exploration and exploitation processes. Finally, the DOLSSO is terminated by the fulfilment of a stop criterion.

Experimental analysis

The experimentation setup and evaluation of results are performed to ensure the effectiveness of the projected system. Later, the obtained outcome of the introduced model is compared with five state-of-art existing metaheuristic algorithms. We have utilized the FogSim simulator in this scheduling to generate the dataset. An energy-efficient open-source tool is used for modelling and simulating resource management in fog/edge computing. The FogSim is integrated with CloudSim to deal

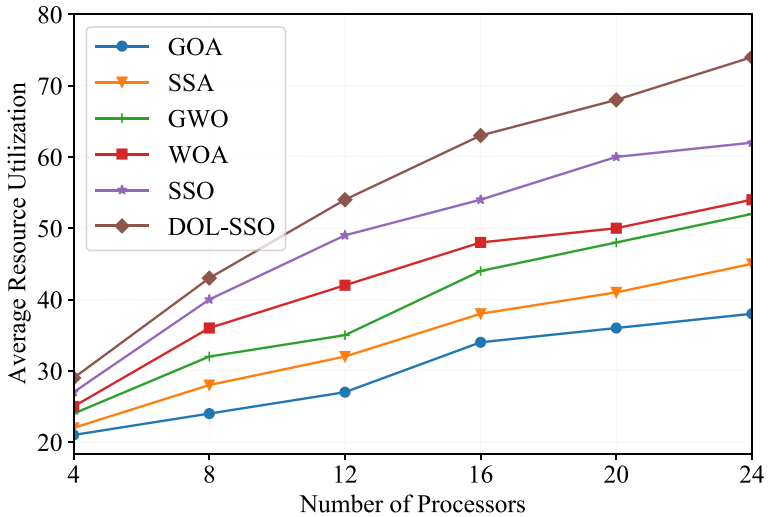

Fig. 3 Resource Consumption concerning the number of CPUs

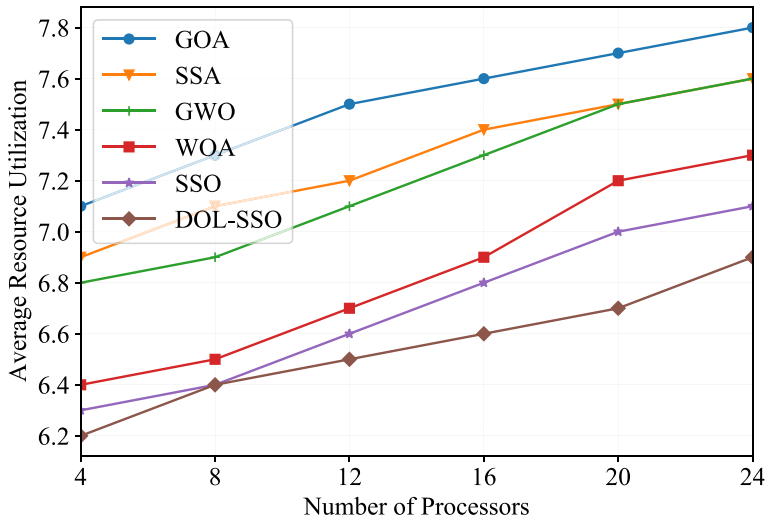

Fig. 4 Average Energy Consumption Ratio concerning the number of CPUs

with the actions between fog computing environments. In CloudSim, different parameters, like data centres, use communication processes for transmission.

Parameter settings

The performance of the SSO algorithm on solving job scheduling is coded using FogSim with CloudSim under Windows 10 on an Intel i5 processor with 3.4GHz and 16GB RAM. The empirical result is compared with other state-of-art metaheuristic algorithms, namely Social Spider Optimization (SSO) [22], grasshopper optimization algorithm (GOA) [39], Salp swarm algorithm (SSA) [40], Grey Wolf Optimization (GWO) [41], and Whale Optimization Algorithm (WOA) [42]. For all experimentation algorithms, the population size, maximum iterations and number of runs are fixed as 60, 100 and 20, respectively. The simulation parameters utilized for this experimentation is illustrated in Table 1.

Result analysis

The dataset is generated with the aid of a FogSim-based simulation tool. All the proposed and compared algorithms are iterated for 100 epochs for each test case with varying tasks, and obtained results are graphically plotted. Further, we experimented on two test cases concerning the number of processors and fog nodes. For the first case, the number of processors varies from 4, 8, 12, 16, 20, and 24 with 20, 40, 80, 160, 320 and 400 tasks, respectively. Each task is allocated to an adequate number of fog servers, and processing orders are determined by solution representation as specified in DOL-based SSO algorithm section. For the second case, the number of fog nodes varies from 5, 10, 15 and 20 with respect to different jobs.

Case 1: experimentation based on the number of processors

In the first case, we have created varying numbers of processors with various tasks. The experimentation results are measured, and the introduced model is contrasted with five metaheuristic algorithms: GOA, SSA, GWO, WOA and SSO. Figure 3 determines the average resource utilization concerning the number of processors. Figure 3 shows that the proposed DOL-SSP algorithm provides better than GOA, SSA, GWO, WOA and SSO. The standard SSA algorithm competes with the proposed algorithm but fails during iterations. At the same time, the proposed DOL-SSO algorithm utilizes the processors effectively by allocating adequate tasks to the available machines. GWO and WOA algorithm

Table 2 Execution time for Jobs

# Fog nodes	GOA	SSA	GWO	WOA	SSO	DOLSSO
5	51.31	52.72	50.45	5.31	51.72	50.15
10	53.87	54.95	52.18	53.14	54.16	51.87
15	56.47	56.39	53.52	53.98	53.78	52.14
20	54.98	55.41	52.49	53.14	52.14	51.89

Fig. 5 Analysis and comparison of Execution time to process job

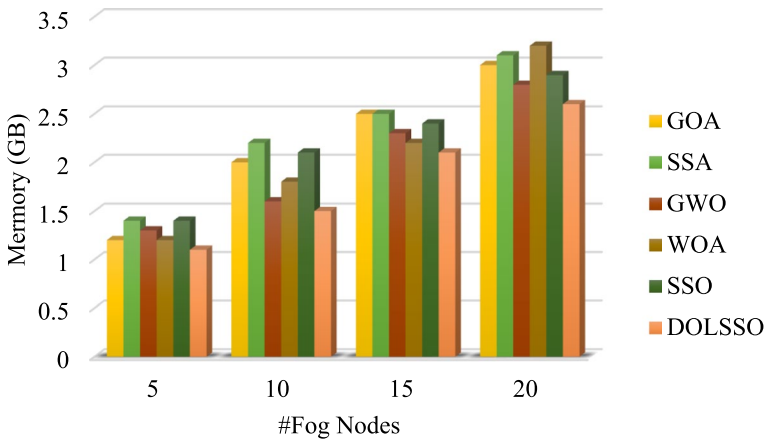

Fig. 6 Analysis and comparison of Allocated memory for the job

Table 3 Allocated memory for jobs

# Fog nodes	GOA	SSA	GWO	WOA	SSO	DOLSSO
5	1.2	1.4	1.3	1.2	1.4	1.1
10	2	2.2	1.6	1.8	2.1	1.5
15	2.5	2.5	2.3	2.2	2.4	2.1
20	3	3.1	2.8	3.2	2.9	2.6

provides moreover similar results in resource utilization. Overall, average resource utilization by DOLSSO was improved on average by 12% more than the standard SSA algorithm.

The average energy depletion ratio concerning the number of processors is represented in Fig. 4. In this experimentation, achieving a lower energy consumption ratio specifies the algorithm hoards the energy and provides better performance. Based on Fig. 4, we noticed that the introduced model gives significant outcomes to all numbers of processors except eight processors compared to other algorithms. Moreover, GWO and SSA provide similar results and attain more energy consumption ratio, which offers less performance than the proposed

method. The standard SSO algorithm and the proposed algorithm achieve the same results due to the random initialization of populations. Also, several ways consume more energy than the proposed algorithm. Though the proposed algorithm attains the same output in 8 processors, the resources are more effectively utilized than the standard SSO algorithm.

Case 2: experimentation based on the number of fog nodes

The efficacy of the proposed model on fog computing scenarios is validated by varying fog nodes from 5 to 20 with a varying number of jobs. CPU clock rate and allocated memory for heterogeneous nodes are taken from [43]. We have considered four cases of fog nodes; the maximum number of iterations is 8000 in all test cases with 20 runs. The introduced model is contrasted with five existing metaheuristic algorithms like GOA, SSA, GWO, WOA and SSO. For evaluation, two performance metrics are utilized: execution time and allocated memory for jobs with respect to completion time. Once the server is distributed with a specific number of jobs, it is locked until it completes its execution. The experimentation of execution time for various jobs concerning fog nodes is observed in Table 2. The execution time for jobs is illustrated in Fig. 5. The table and Fig. 5 clearly show that the introduced model provides improved results concerning minimum execution time than the other compared algorithms.

The allocated memory for jobs concerning the number of fog nodes is observed in Table 2. The pictorial representation of allocated memory for jobs is illustrated in Fig. 6. As observed from the results, Table 3 and Fig. 6 specifies that the concert of the DOLSSO method outperforms well than compared algorithms. The maximum allocated memory concerning 20 fog nodes for various determined jobs attained by DOLSSO (2.6 GB) is lesser than the standard SSO (2.9 GB). In addition, allocated memory for jobs achieved by GWO (2.8 GB) and SSO (2.9 GB) are closer. Based on the experimentation results, incorporating the DOL strategy into SSO improves performance by eradicating local optimal struck and a better convergence rate.

Conclusion

In the last few years, fog computing has given great attention to researchers, industrialists and the community due to its computational services. We addressed the job scheduling issue in the fog computing setting with reduced CPU time utilization and memory usage. This work introduces a novel version of the SSO method by incorporating the dynamic opposition learning (DOL) approach, namely the DOLSSO algorithm. The proposed model enriches the solution quality by eradicating the local optimal struck and boosting the convergence rate towards the optimal solution. The experimentation is carried out in the FogSim simulation tool with two different scenarios. The first scenario pacts with several processors concerning the number of tasks, and the second test case deals with the number of fog nodes concerning the number of jobs. The proposed infrastructure guarantees the execution of the data request and satisfies mobile users effectively using the DOLSSO algorithm. The empirical result shows the algorithm's effectiveness in obtaining an optimal schedule in a Fog computing environment. The results show that the proposed method achieves ~10% - 15% better CPU utilization and ~5%-10% less energy consumption than other algorithms. Further, this work can be extended to handle the multi-objective flexible job scheduling issue by incorporating self-adaptive parameters into the DOLSSO algorithm.

Acknowledgements
This research was supported by the Taif University Researchers Supporting Project number (TURSP-2020/231), Taif University, Taif, Saudi Arabia.

Authors' contributions
Conceptualization - Kuppusamy P; methodology- Kuppusamy P, N. Marline Joys Kumari; validation- Rajakumar Ramalingam, Mamoon Rashid, Abdul Rehman Javed; formal analysis- Wael Y. Alghamdi, Hashem Alyami; writing—original draft preparation- Kuppusamy P; writing—review and editing- Mamoon Rashid, Abdul Rehman Javed; supervision- Rajakumar Ramalingam; funding acquisition- Wael Y. Alghamdi. All authors have read and agreed to the submitted version of the manuscript.

Funding
Researchers Supporting Project number (TURSP-2020/231), Taif University, Taif, Saudi Arabia.

Availability of data and materials
The data that support the findings of this study are available from the first author upon reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no conflict of interest.

Author details
[1]School of Computer Science & Engineering, VIT-AP University, Amaravati, Andhra Pradesh, India. [2]Department of CSE, Anil Neerukonda Institute of Technology and Sciences, Sanghivalasa, Bheemunipatnam, India. [3]Department of Computer Science, College of Computers and Information Technology, Taif University, P.O.Box 11099, Taif 21944, Saudi Arabia. [4]Department of CST, Madanapalle Institute of Technology & Science, Madanapalle, India. [5]Department of Cyber Security, Air University, Islamabad 44000, Pakistan. [6]Department of Electrical and Computer Engineering, Lebanese American University, Byblos, Lebanon. [7]Department of Computer Engineering, Faculty of Science and Technology, Vishwakarma University, Pune 411048, India.

Received: 25 August 2022 Accepted: 7 December 2022
Published online: 23 December 2022

References

1. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Futur Gener Comput Syst 25(6):599–616
2. Eleyan A, Eleyan D (2015) Forensic process as a service (FPaaS) for cloud computing. In: Intelligence and security informatics conference (EISIC), 2015 European. IEEE, pp 157–160
3. The Network.Cisco Delivers Vision of Fog Computing to Accelerate Value from Billions of Connected Devices. http://newsroom.cisco.com/press-release-content?articleId=1334100.M
4. Deng R, Rongxing L, Lai C, Luan TH, Liang H (2016) Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J 3(6):1171–1181
5. Şahman MA (2021) A discrete spotted hyena optimizer for solving distributed job shop scheduling problems. Appl Soft Comput 106:107349
6. Naderi B, Azab A (2014) Modeling and heuristics for scheduling of distributed job shops. Expert Syst Appl 41(17):7754–7763
7. Gao J, Chen R (2011) A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. Int J Comput Intell Syst 4(4):497–508
8. Xie J, Gao L, Pan Q-k, Fatih Tasgetiren M (2019) An effective multi-objective artificial bee colony algorithm for energy efficient distributed job shop scheduling. Proc Manufact 39:1194–1203
9. Lei D (2008) Pareto archive particle swarm optimization for multi-objective fuzzy job shop scheduling problems. Int J Adv Manuf Technol 37(1):157–165
10. Rossi A, Dini G (2007) Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. Robot Comput Integr Manuf 23(5):503–516
11. Singh S, Kurmi J, Tiwari SP (2015) A hybrid genetic and cuckoo search algorithm for job scheduling. Int J Sci Res Publ 5(6):1–4
12. Wang L, Pan Q-K, Suganthan PN, Wang W-H, Wang Y-M (2010) A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput Oper Res 37(3):509–520
13. Lu C, Gao L, Pan Q, Li X, Zheng J (2019) A multi-objective cellular grey wolf optimizer for hybrid flowshop scheduling problem considering noise pollution. Appl Soft Comput 75:728–749
14. Liu M, Yao X, Li Y (2020) Hybrid whale optimization algorithm enhanced with Lévy flight and differential evolution for job shop scheduling problems. Appl Soft Comput 87:105954
15. Atay Y, Kodaz H (2014) Implementation of CSA with clone-mutation mechanism to the JSSP. Int J Mach Learn Comput 4(1):6
16. Bezdan T, Zivkovic M, Antonijevic M, Zivkovic T, Bacanin N (2021) Enhanced flower pollination algorithm for task scheduling in cloud computing environment. In: Machine learning for predictive analysis. Springer, Singapore, pp 163–171
17. Yadav AM, Tripathi KN, Sharma SC (2022) A bi-objective task scheduling approach in fog computing using hybrid fireworks algorithm. J Supercomput 78(3):4236–4260
18. Najafizadeh A, Salajegheh A, Rahmani AM, Sahafi A (2022) Multi-objective task scheduling in cloud-fog computing using goal programming approach. Clust Comput 25(1):141–165
19. Wu X, Shen X, Zhao N, Shaomin W (2020) An improved discrete pigeon-inspired optimisation algorithm for flexible job shop scheduling problem. Int J Bio-Inspir Comput 16(3):181–194
20. Cuevas E, Cienfuegos M, Zaldívar D, Pérez-Cisneros M (2013) A swarm optimization algorithm inspired in the behavior of the social-spider. Expert Syst Appl 40(16):6374–6384
21. Baş E, Ülker E (2020) A binary social spider algorithm for continuous optimization task. Soft Comput 24(17):12953–12979
22. Cuevas E, Cienfuegos M (2014) A new algorithm inspired in the behavior of the social-spider for constrained optimization. Expert Syst Appl 41(2):412–425
23. Mahato DP, Singh RS (2018) On maximizing reliability of grid transaction processing system considering balanced task allocation using social spider optimization. Swarm Evol Comput 38:202–217
24. Zhou Y, Zhou Y, Luo Q, Abdel-Basset M (2017) A simplex method-based social spider optimization algorithm for clustering analysis. Eng Appl Artif Intell 64:67–82
25. Ouadfel S, Taleb-Ahmed A (2016) Social spiders' optimization and flower pollination algorithm for multi-level image thresholding: a performance study. Expert Syst Appl 55:566–584
26. Shao Z, Zhuge Q, Xue C, Sha EH-M (2005) Efficient assignment and scheduling for heterogeneous dsp systems. IEEE Trans Parallel Distrib Syst 16(6):516–525
27. Ningning S, Chao G, Xingshuo A, Qiang Z (2016) Fog computing dynamic load balancing mechanism based on graph repartitioning. China Commun 13(3):156–164
28. Cardellini V, Grassi V, Presti FL, Nardelli M (2015) On QoS-aware scheduling of data stream applications over fog computing infrastructures. In: Computers and communication (ISCC), 2015 IEEE symposium on. IEEE, Larnaca, p 271–76.
29. Oueis J, Strinati EC, Barbarossa S (2015) The fog balancing: load distribution for small cell cloud computing. In: Vehicular technology conference (VTC spring), 2015 IEEE 81st. IEEE, Glasgow, p 1–6.
30. den Bossche V, Ruben KV, Broeckhove J (2011) Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In: Cloud computing technology and science (CloudCom), 2011 IEEE third international conference on. IEEE, Athens, p 320–27.
31. Zeng L, Veeravalli B, Li X (2012) Scalestar: budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In: Advanced information networking and applications (AINA), 2012 IEEE 26th international conference on. IEEE, Fukuoka, p 534–41.
32. Fleming PJ, Zalzala AMS (editor) (1997) Genetic algorithms in engineering systems. Institution of Electrical Engineers, Herts
33. Naderi B, Azab A (2015) An improved model and novel simulated annealing for distributed job shop problems. Int J Adv Manuf Technol 81(1):693–703
34. Tang H, Fang B, Liu R, Li Y, Guo S (2022) A hybrid teaching and learning-based optimization algorithm for distributed sand-casting job-shop scheduling problem. Appl Soft Comput 120:108694
35. Li H, Wang X, Peng J (2022) A hybrid differential evolution algorithm for flexible job shop scheduling with outsourcing operations and job priority constraints, Expert Systems with Applications, p 117182, ISSN 0957-4174. https://doi.org/10.1016/j.eswa.2022.117182
36. Aggarwal S, Chatterjee P, Bhagat RP, Purbey KK, Nanda SJ (2018) A social spider optimization algorithm with chaotic initialization for robust clustering. Proc Comput Sci 143:450–457
37. Xu Y, Yang Z, Li X, Kang H, Yang X (2020) Dynamic opposite learning enhanced teaching–learning-based optimization. Knowl-Based Syst 5(188):104966
38. Zhang G, Gao L, Shi Y (2011) An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 38(4):3563–3573
39. Saremi S, Mirjalili S, Lewis A (2017) Grasshopper optimisation algorithm: theory and application. Adv Eng Softw 105:30–47
40. Mirjalili S, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM (2017) Salp swarm algorithm: a bio-inspired optimizer for engineering design problems. Adv Eng Softw 114:163–191
41. Natesan G, Chokkalingam A (2019) Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm. ICT Express 5(2):110–114
42. Jiang T, Zhang C, Zhu H, Jiuchun G, Deng G (2018) Energy-efficient scheduling for a job shop using an improved whale optimization algorithm. Mathematics 6(11):220
43. Bitam S, Zeadally S, Mellouk A (2017) Fog computing job scheduling optimization based on bees swarm. Enterp Inf Syst 12,(4):1–25.

Publisher's Note