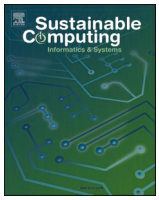
Sustainable Computing: Informatics and Systems 43 (2024) 101012

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom



Improved synergistic swarm optimization algorithm to optimize task scheduling problems in cloud computing

Laith Abualigah [a,b,c,*], Ahmad MohdAziz Hussein [d], Mohammad H. Almomani [e], Raed Abu Zitar [f], Hazem Migdady [g], Ahmed Ibrahim Alzahrani [h], Ayed Alwadain [h]

[a] Computer Science Department, Al al-Bayt University, Mafraq 25113, Jordan
[b] Applied science research center, Applied Science Private University, Amman 11931, Jordan
[c] Jadara Research Center, Jadara University, Irbid 21110, Jordan
[d] Department of Computer Science, Faculty of Information Technology, Middle East University, Amman, Jordan
[e] Department of Mathematics, Facility of Science, The Hashemite University, P.O box 330127, Zarqa 13133, Jordan
[f] Sorbonne Center of Artificial Intelligence, Sorbonne University, Paris, France
[g] CSMIS Department, Oman College of Management and Technology, 320 Barka, Oman
[h] Computer Science Department, Community College, King Saud University, Riyadh, 11437, Saudi Arabia

ARTICLE INFO

Keywords:
Cloud Computing
Task Scheduling
Jaya Algorithm
Synergistic Swarm Optimization
Levy Flight Mechanism
Resource Utilization

ABSTRACT

Cloud computing has emerged as a cornerstone technology for modern computational paradigms due to its scalability and flexibility. One critical aspect of cloud computing is efficient task scheduling, which directly impacts system performance and resource utilization. In this paper, we propose an enhanced optimization algorithm tailored for task scheduling in cloud environments. Building upon the foundation of the Jaya algorithm and Synergistic Swarm Optimization (SSO), our approach integrates a Levy flight mechanism to enhance exploration-exploitation trade-offs and improve convergence speed. The Jaya algorithm's ability to exploit the current best solutions is complemented by the SSO's collaborative search strategy, resulting in a synergistic optimization framework. Moreover, the incorporation of Levy flights injects stochasticity into the search process, enabling the algorithm to escape local optima and navigate complex solution spaces more effectively. We evaluate the proposed algorithm against state-of-the-art approaches using benchmark task scheduling problems in cloud environments. Experimental results demonstrate the superiority of our method in terms of solution quality, convergence speed, and scalability. Overall, our proposed Improved Jaya Synergistic Swarm Optimization Algorithm offers a promising solution for optimizing TSCC (TSCC), contributing to enhanced resource utilization and system performance in cloud-based applications. The proposed method got 88 % accuracy overall and 10 % enhancement compared to the original method.

1. Introduction

Cloud computing has revolutionized the way computational resources are provisioned and utilized, offering unprecedented scalability, flexibility, and cost-effectiveness for various applications and services [1,2]. At the core of cloud computing lies efficient resource management, particularly in the scheduling of tasks across distributed and virtualized infrastructure [3,4]. Task scheduling plays a pivotal role in optimizing resource utilization, minimizing execution time, and enhancing overall system performance [5,6]. However, the inherent complexities and dynamic nature of cloud environments pose significant challenges for traditional scheduling algorithms [7,8].

In recent years, metaheuristic optimization techniques have gained prominence for addressing the intricate task scheduling problem in cloud computing [8,9]. These algorithms offer a promising avenue for finding near-optimal solutions in large-scale, dynamic, and uncertain environments [10–12]. Among these techniques, the Jaya algorithm and Synergistic Swarm Optimization (SSO) have demonstrated effectiveness in solving various optimization problems [13,14]. The Jaya algorithm, inspired by the concept of natural selection, iteratively improves candidate solutions by exploiting the current best solution without relying on any explicit parameter tuning. On the other hand, SSO harnesses the collective intelligence of a swarm of particles to explore the solution space collaboratively, enabling efficient search and adaptation

* Corresponding author at: Computer Science Department, Al al-Bayt University, Mafraq 25113, Jordan,.
E-mail address: aligah.2020@gmail.com (L. Abualigah).

https://doi.org/10.1016/j.suscom.2024.101012
Received 22 December 2023; Received in revised form 7 June 2024; Accepted 13 June 2024
Available online 18 June 2024

[15].

Despite their merits, both the Jaya algorithm and SSO exhibit certain limitations when applied to complex optimization problems such as TSCC. The Jaya algorithm may struggle with balancing exploration and exploitation, leading to premature convergence or suboptimal solutions in rugged search spaces. Similarly, SSO's reliance on swarm dynamics may encounter difficulties in effectively exploring diverse regions of the solution space, particularly in high-dimensional and multimodal optimization landscapes [15,16].

To address these challenges, we propose an Improved Jaya Synergistic Swarm Optimization Algorithm for Task Scheduling Optimization in Cloud Computing. Our approach aims to leverage the strengths of both algorithms while mitigating their weaknesses through a synergistic integration framework. By incorporating a Levy flight mechanism into the optimization process, we enhance the algorithm's exploration capabilities, enabling it to escape local optima and navigate complex solution spaces more effectively. The Levy flight mechanism introduces stochasticity into the search process, facilitating a more robust exploration-exploitation trade-off and improving convergence speed.

In this paper, we present a comprehensive study on the development and evaluation of our proposed algorithm. We compare its performance against state-of-the-art approaches using benchmark task scheduling problems in cloud environments. Through extensive experimentation and analysis, we demonstrate the effectiveness and superiority of our method in terms of solution quality, convergence speed, and scalability. Ultimately, our proposed Improved Jaya Synergistic Swarm Optimization Algorithm offers a promising solution for optimizing TSCC, contributing to enhanced resource utilization and system performance in cloud-based applications. The main contributions of this paper are given as follows.

- We introduce a novel optimization algorithm tailored specifically for task scheduling optimization in cloud computing. Our algorithm combines the strengths of the Jaya algorithm and Synergistic Swarm Optimization (SSO) while addressing their limitations through a synergistic integration framework.
- To enhance exploration capabilities and improve convergence speed, we incorporate a Levy flight mechanism into the optimization process. This stochastic search strategy enables the algorithm to efficiently explore diverse regions of the solution space, facilitating the discovery of high-quality solutions in complex and dynamic cloud environments.
- We conduct extensive experiments to evaluate the performance of our proposed algorithm against state-of-the-art approaches using benchmark task scheduling problems in cloud environments. Through rigorous analysis, we demonstrate the effectiveness and superiority of our method in terms of solution quality, convergence speed, and scalability.
- Our proposed Improved Jaya Synergistic Swarm Optimization Algorithm offers a promising solution for addressing the task scheduling optimization challenge in cloud computing. By enhancing resource utilization and system performance, our algorithm contributes to the advancement of cloud-based applications and services.

In this paper, we present a structured investigation into task scheduling optimization in cloud computing, outlining our proposed algorithm and its evaluation against existing approaches. The paper is organized as follows: In Section 2, we provide an overview of related works in the field of task scheduling optimization in cloud computing. We discuss various metaheuristic optimization techniques and their applications, highlighting the strengths and limitations of existing approaches. Section 3 details our proposed method, the Improved Jaya Synergistic Swarm Optimization Algorithm, including the integration of the Levy flight mechanism and the synergistic framework combining the Jaya algorithm and Synergistic Swarm Optimization. We elucidate the algorithmic design, optimization process, and key components contributing to its effectiveness in addressing task scheduling challenges in cloud environments. Subsequently, in Section 4, we present the results of comprehensive experiments conducted to evaluate the performance of our proposed algorithm. We provide details of the experimental setup, including benchmark task scheduling problems and parameter settings. We analyze the obtained results, comparing them with state-of-the-art approaches to assess the superiority of our method in terms of solution quality, convergence speed, and scalability. Finally, in Section 5, we conclude the paper by summarizing our findings and highlighting avenues for future research. We discuss the implications of our proposed algorithm in enhancing resource utilization and system performance in cloud-based applications, and we outline potential directions for further refinement and extension of the proposed approach.

2. Related works

How tasks are organized in the cloud has a significant impact on how resources are used and how much it costs to run the business [17–20]. Many metaheuristic algorithms and variants have been suggested to simplify scheduling procedures in order to increase the operational efficiency of job execution in cloud settings [21,22]. The most related works are given as follows.

Recent research has focused on cloud task scheduling [23,24]. Effectively scheduling massive user-submitted activities in cloud settings boosts firms' competitiveness and economic performance. This research examines cloud task scheduling and presents PSOPGA, a particle swarm optimization genetic hybrid algorithm based on phagocytosis, to meet the demand for an effective scheduling method in real-world circumstances [25]. Each generation of the particle swarm divides, and the genetic algorithm's phagocytosis process and crossover mutation change particle placements in the sub-population, expanding the solution space. The subpopulations are then blended to maintain particle population diversity and reduce algorithm local optima. Finally, a feedback mechanism relays the particle's and its counterparts' flight experiences to the next generation's particle population, assuring optimum solutions. Simulations show that the proposed technique improves cloud job completion time and convergence accuracy compared to various other strategies. The technique performs well in cloud job scheduling.

By using a multiobjective optimization strategy to improve cloud system performance with given computing resources, the whale optimization algorithm (WOA) is proposed for scheduling tasks in the cloud [26]. Enhanced WOA for Cloud Task Scheduling (EWC) is a novel approach that builds on this basis and enhances the search for optimum solutions capabilities of the WOA-based technique. They outline EWC's full implementation, and simulation-driven studies show that EWC finds optimum task scheduling methods faster and more precisely than previous metaheuristic algorithms. Furthermore, EWC shows improved efficiency in the use of system resources for a variety of jobs, from little to large.

When it comes to cloud computing, one of the biggest obstacles is efficient job scheduling. It is quite challenging to find the best solution to the NP-complete issue of task scheduling, especially when working with large quantities of tasks. When working in the cloud, it's important to minimize makespan and maximize resource usage for effective job scheduling across several virtual machines. In this paper, they provide a new method for scheduling tasks in cloud computing settings that takes into account many objectives at once [27]. This method is called the hybrid antlion optimization algorithm with elite-based differential evolution. The suggested solution, MALO, takes into account the fact that the issue has several objectives and aims to reduce makespan while maximizing resource consumption concurrently. To improve exploitation capabilities and avoid being trapped in local optima, MALO uses the antlion optimization algorithm and augments it with elite-based differential evolution as a local search method. Using the CloudSim toolbox,

they ran two sets of tests using both simulated and actual trace datasets. The results showed that MALO is better than other optimization algorithms; it is especially useful for vast search areas since it converges faster, making it ideal for solving complex scheduling issues. In addition, statistical t-tests performed on the findings confirmed that MALO significantly improved performance.

This study introduces a cloud-optimized ant colony optimization algorithm to address issues like uneven workload distribution, slow convergence rates, and underutilization of virtual machine resources in previous task scheduling optimization approaches [28]. Using cloud computing job scheduling insights, they create a scheduling model using an enhanced ant colony method to avoid local optimization issues. Next, a task scheduling satisfaction function is created to find the best task scheduling solution by decreasing waiting time, improving resource load balancing, and reducing task completion costs. They also use reward and punishment coefficients to improve the ant colony algorithm's pheromone update rules, speeding solution convergence. Dynamic volatility coefficient updates improve strategy performance, and local pheromone updating includes virtual machine load weight coefficients for load equilibrium. Experiments with Cloudsim prove the proposed method works. They found that the proposed approach has the fastest convergence, shortest completion times, most balanced task distribution, and maximum virtual machine resource usage rates. Thus, our job scheduling optimization technique excels in cloud computing.

Cloud computing solutions have been popular since their creation because they provide a common infrastructure where customers may obtain customized services without worrying about location or delivery, paying only for the services used. Despite their benefits, cloud computing systems struggle with scheduling and energy management. Given the different customers and services in these systems, good scheduling is vital to reduce provider and consumer costs and optimize energy use. They present a two-step hybrid task scheduling strategy that includes energy and time restrictions to overcome this difficulty [29]. First, prioritize tasks, then allocate them to processors. The Energy-Conscious Scheduling Heuristic, an energy-aware model, assigns tasks to processors after task ranking and main chromosome creation. The simulation results show that the suggested algorithm optimizes work scheduling while considering energy consumption better than competing techniques.

Cloud computing stands as a proficient technology catering to the demands of big data applications. The optimization of cloud system makespan while enhancing resource utilization is imperative for cost reduction. Task scheduling poses a formidable challenge in meeting these requirements, necessitating both efficacy and efficiency. To address this challenge, this paper introduces a task scheduler featuring discrete variants of the particle swarm optimization (PSO) algorithm tailored for cloud computing task scheduling [30]. To evaluate effectiveness, these methods are juxtaposed with three well-known heuristic algorithms for task scheduling dilemmas. Empirical findings highlight the efficiency and efficacy of the suggested methodologies. Especially for the scheduler based on PSO introduced here, the utilization of a logarithmic declining tactic emerges as the most advantageous in delivering an ideal scheduling arrangement. The average makespan of the PSO-based scheduler employing the logarithmic decreasing strategy is observed to decrease by 19.12 %, 21.42 %, and 15.14 % relative to the gravitational search algorithm, artificial bee colony algorithm, and dragonfly algorithm, respectively.

Cloud computing is an emerging distributed technology that provides low-cost, dynamically scalable computer resources. TSCC is crucial to system performance and customer satisfaction. Despite several task scheduling techniques, most focus on lowering completion time and ignoring burden balance. Existing methods for QoS management may be improved. MGGS (modified genetic algorithm (GA) paired with greedy technique) is introduced in this paper. MGGS optimizes task scheduling using a modified GA algorithm and greedy approach. MGGS achieves optimum solutions with fewer iterations, unlike other algorithms. To evaluate MGGS, they compare its performance to other algorithms using metrics like total completion time, average response time, and QoS parameters [31]. Experimental data show that MGGS outperforms other work scheduling methods.

Effective cloud computing resource utilization requires efficient job scheduling. Task scheduling is NP-hard because it must be done across several virtual machines while reducing makespan and optimizing resource consumption, especially in Big Data applications. An intelligent hybrid Dragonfly Algorithm is used to schedule Big Data tasks for IoT cloud computing applications in this research [32]. MHDA uses the Dragonfly algorithm, a new optimization method inspired by dragonflies' swarming. Multiobjective MHDA reduces makespan and improves resource usage. By using β-hill climbing as a local exploratory search mechanism, the Dragonfly Algorithm may better exploit local optima and reduce the chance of entrapment. MHDA's performance is compared to various task scheduling algorithms in two CloudSim toolkit experiments using synthetic and real trace datasets. MHDA outperforms other algorithms in convergence rates and outcomes by 17.12 %, according to analytical assessments such as t-tests. MHDA's effectiveness in Big Data job scheduling difficulties supports its practical use.

TSCC is difficult due to different cloudlets, deadline limitations across hybrid cloud resources, and varied quality criteria. The cloud computing job scheduling problem is addressed by this study [33]. Chemical Reaction Partial Swarm Optimization is a new hybrid job scheduling approach. This approach hybridizes classical chemical reaction optimization and partial swarm optimization to improve job allocation among virtual machines. The method optimizes schedule sequencing to process tasks based on demand and deadline simultaneity, improving quality metrics like cost, energy, and makespan. The algorithm's usefulness is shown via a CloudSim toolkit simulation experiment. Comparative investigations show a 1–6 % decrease in execution time across different virtual machines and job numbers, with some gains surpassing 10 %. Results for makespan show algorithmic efficacy of 5–12 %, overall cost of 2–10 %, and energy consumption rates of 1–9 %.

Various research efforts have addressed the challenges of TSCC, recognizing its pivotal role in optimizing system performance and resource utilization. These studies have introduced innovative methodologies and algorithms tailored to the complexities of cloud environments. For instance, PSOPGA utilizes a hybrid particle swarm optimization genetic algorithm to enhance scheduling accuracy, while EWC and MALO leverage multiobjective optimization strategies to improve efficiency and convergence rates. Additionally, approaches like the cloud-optimized ant colony optimization algorithm and MGGS employ modified genetic algorithms and greedy techniques to optimize task allocation and scheduling. The introduction of novel optimization algorithms, such as the Dragonfly Algorithm and Chemical Reaction Partial Swarm Optimization, further enhances scheduling effectiveness, particularly in addressing the demands of Big Data applications and diverse quality criteria. Through simulations and comparisons with existing methods, these studies demonstrate significant improvements in makespan reduction, resource utilization, and overall system performance, highlighting the importance of efficient task scheduling in advancing cloud computing capabilities and meeting diverse user needs. Table 1 summarize the studies by their main focus, proposed method, evaluation techniques, and key results, providing a clear and detailed overview.

3. The proposed JSSOA method

3.1. Procedure of synergistic swarm optimization algorithm

Following is a rundown of the main steps involved in the proposed Synergistic Swarm Optimization Algorithm (SSOA) [14]. Eq. (1) defines the starting point for the optimization trip, which is the stochastic internalization of possible solutions.

Table 1
An overview of the given studies.

Study	Main Focus	Proposed Method	Evaluation Techniques	Key Results
[25]	Cloud task scheduling using a hybrid algorithm to improve operational efficiency.	PSOPGA (Particle Swarm Optimization Genetic Hybrid Algorithm based on Phagocytosis)	Simulations	Improved job completion time and convergence accuracy compared to other strategies.
[26]	Multiobjective optimization for cloud system performance improvement.	Enhanced Whale Optimization Algorithm (EWC)	Simulation-driven studies	EWC finds optimal task scheduling methods faster and more precisely, improves resource efficiency.
[27]	Efficient job scheduling considering multiple objectives.	Hybrid Antlion Optimization Algorithm with Elite-Based Differential Evolution (MALO)	CloudSim toolbox tests with simulated and real datasets	MALO converges faster, better performance in complex scheduling problems, statistically significant improvements.
[28]	Addressing issues in task scheduling such as workload distribution and convergence rates.	Cloud-Optimized Ant Colony Optimization Algorithm	Experiments with CloudSim	Fastest convergence, shortest completion times, balanced task distribution, maximum VM resource usage.
[29]	Energy and time constraints in task scheduling to reduce costs and optimize energy use.	Two-step Hybrid Task Scheduling Strategy with Energy-Conscious Scheduling Heuristic	Simulations	Better optimization of work scheduling considering energy consumption compared to competing techniques.
[30]	Optimization of makespan and resource utilization in cloud task scheduling.	Discrete Particle Swarm Optimization (PSO) Algorithm	Comparison with heuristic algorithms	PSO-based scheduler reduces makespan significantly compared to other algorithms.
[31]	Balancing task completion time and workload in cloud computing.	Modified Genetic Algorithm with Greedy Technique (MGGS)	Performance metrics (completion time, response time, QoS)	MGGS outperforms other scheduling methods in terms of completion time and response time.
[32]	Efficient scheduling for Big Data tasks in IoT cloud computing.	Intelligent Hybrid Dragonfly Algorithm (MHDA)	CloudSim toolkit experiments	MHDA shows better convergence rates and outcomes, significant performance improvements.
[33]	Addressing TSCC with varied quality criteria and hybrid cloud resources.	Chemical Reaction Partial Swarm Optimization	CloudSim toolkit simulation	Decrease in execution time, improved makespan, overall cost, and energy consumption rates.

$$X = rand(N, Dim) \quad .*(UB - LB) + LB \tag{1}$$

Formula (1) constructs a matrix X, with dimensions (N x D), comprising randomly selected values bounded within a specified range [14]. The format of this Matrix is illustrated in (2).

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,Dim} \\ \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,Dim} \end{bmatrix} \tag{2}$$

In this context, N denotes the number of particles or solutions, while D signifies the number of dimensions or variables pertinent to the problem at hand. UB and LB denote vectors signifying the upper and lower bounds, correspondingly, for each dimension within the problem space. Eq. (3) is utilized to refine current solutions (X) [14].

$$Xnew(i,j) = \quad X(i,j) + \quad v(i,j) \tag{3}$$

In this context, Xnew(i,j) symbolizes the fresh, optimized position j of the ith solution. In contrast, X(i,j) denotes the position j of the ith solution, and v(i,j) represents the value of position j of the ith solution. Additionally, in conjunction with the velocity update formula, a dynamic attraction equation is presented to guide the particles towards more advantageous areas within the exploration domain. This equation is designed to dynamically direct the particles by taking into account the attractiveness of both local and global positions. Its formulation is outlined in Eq. (4).

$$v_{new}(i,j) = IWV + PBC + GBC + DAC + ANIC + MDC \tag{4}$$

The calculations for the value of vnew (i,j) are defined by the following equations. The next way to determine the IWV is by using the following formula [14].

$$IWV = w(t) \quad * \quad v(i,j) \tag{5}$$

A method for adaptive regulation of the exploration-exploitation equilibrium is provided by the variable "w" in relation to the inertia weight parameter (w). Moreover, the adaptive neighborhood interaction equation enhances a concentrated exploration of the search space by assigning greater importance to particles with superior fitness levels, facilitating a more efficient convergence of the swarm. Particles demonstrating higher fitness levels can exert more influence on the movements of neighboring particles through an equation that modifies the intensity of interactions based on their fitness levels. Employing an adaptive equation, the inertia weight can be adjusted with each iteration, such as:

$$w(t+1) = \quad w(t) * \quad (1 \quad - \exp(-k \quad * \quad t)) \tag{6}$$

In this context, "t" denotes an ongoing iteration, while "k" remains a constant dictating the rate at which the inertia weight diminishes. As iterations progress, the approach transitions from exploration to exploitation, refining its search and converging towards the optimal solution by reducing the inertia weight [14]. The following outlines the procedure for calculating the personal best coefficient (PBC).

$$PBC = \quad r1 \quad * \quad (eps * rand(pbest) \quad - \quad X_i) \tag{7}$$

In this scenario, "r1" represents a randomly generated value, "eps" denotes a minute value, "rand(pbest)" signifies a randomly chosen solution from the existing candidate solutions, and "Xi" refers to solution number i. The computation of the global best coefficient (GBC) is depicted as follows.

$$GBC = \quad r2 * gbest_t \quad - \quad X_i \tag{8}$$

In this context, "r2" denotes a pseudo-random integer, "gbest(t)" represents the best global solution discovered thus far at iteration t, and "Xi" signifies solution i. To promote exploration across diverse regions within the search space, an equation is employed to sustain variability. The following outlines the procedure for computing the DAC, or dynamic attraction coefficient [14].

$$DAC = \quad r3 * \frac{attract_i}{c1} - \quad X_i \tag{9}$$

In this context, "r3" signifies a randomly generated value, "attract(i)" denotes the position exhibiting the highest local attraction value within the vicinity of the ith particle, "c1" represents an additional acceleration coefficient for the dynamic attract term, and "Xi" refers to solution number i. The dynamic attraction term directs particles toward exceedingly attractive positions, facilitating expedited convergence toward optimal solutions. The calculation of the adaptive neighborhood interaction coefficient (ANIC) is delineated as follows.

$$ANIC = r4 * rand(bestf) \quad - bestf_i \tag{10}$$

In this scenario, "r4," "rand(bestf)," and "bestf(i)" denote the fitness values of the current fitness solutions and a randomly generated value,

respectively. The following outlines the process for determining the diversity maintenance coefficient (DMC) [14].

$$DMC = r5 * \frac{diversity_i}{c2} - \quad X_i \tag{11}$$

Here, "r5" stands for a number that is created at random, and "c2" is an extra acceleration coefficient for the diversity component. "diversityi" indicates a swarm location that optimizes diversity near the ith particle.

To keep the swarm's solution variety intact and prevent early convergence, the diversity term promotes the exploration of less-explored locations. These novel equations improve the SSO method by including adaptive processes like diversity maintenance, inertia weight adaptation, adaptive neighborhood interactions, and dynamic attraction. The algorithm is able to fine-tune its behavior and concentration after implementing these improvements. First, we have the Synergistic Swarm Optimization Algorithm (SSOA) main process. The main procedure of SSOA is given in Algorithm 1.

Algorithm 1. Synergistic Swarm Optimization Algorithm (SSOA)

Initialization:
- Generate random positions for each particle in the swarm
- Initialize velocities to zero
- Set current-best positions to initial positions
- Set the global best position as empty
- Initialize personal best fitness values to infinity
- Set global best fitness value to infinity

Main Loop:
For t = 1 to T do
- Evaluate fitness for each particle
- Update personal best positions
- Update global best position
- Update inertia weight

For each particle in the swarm, do
- Generate random numbers r1, r2, r3, r4, r5
- Update velocity:
  Update velocity using the novel equations incorporating c1, c2, r1, r2, r3, r4.
- Update position:
  Update position based on the updated velocity.
- Apply boundary constraints (if any) to the position

End for
End for

Results:
- Display global best position
- Display global best fitness

Fitness Evaluation Function:
- Evaluate fitness for each particle (modify according to your problem)
- Calculate fitness for the particle at position positions (i, :)
- Replace it with your fitness evaluation code.

Apply Boundary Constraints Function:
- Apply constraints to the position if needed
- Modify the position to satisfy any constraints of the optimization problem

3.2. Procedure of Jaya algorithm

The Jaya Algorithm is a population-based optimization technique inspired by the natural concept of evolution [34]. It belongs to the class of metaheuristic algorithms, specifically targeting continuous optimization problems. Developed by R.V. Rao, the algorithm is renowned for its simplicity and effectiveness in finding optimal or near-optimal

solutions across various domains [35].

In engineering and optimization problems, finding the global optimum can be challenging due to the presence of complex, high-dimensional search spaces and non-linear objective functions. Traditional optimization techniques often struggle with convergence to global optima and may be trapped in local optima. The Jaya Algorithm was conceived as a robust and efficient optimization approach that could overcome these challenges by emulating the principles of natural selection. The main procedure of the Jaya Algorithm is given in Algorithm 2.

Algorithm 2. Jaya Algorithm

Initialization:

- Initialize the population of candidate solutions randomly within the solution space.

Evaluate Fitness:

- Evaluate the fitness or objective function value of each candidate solution.

Update Solutions:

- Improve solutions iteratively by comparing each pair of candidate solutions.
- For each dimension of a candidate solution, update its value based on the better solution among the pair using the formula:
- Update: x_new = x_old + r * (x_best - x_worst)
- Where x_old is the old value of the dimension, x_best is the corresponding dimension of the best solution, x_worst is the corresponding dimension of the worst solution, and r is a random number between 0 and 1.

Termination:

- Repeat the updating process until a termination criterion is met, such as reaching a maximum number of iterations or achieving a satisfactory solution.

The Jaya Algorithm is a population-based optimization technique inspired by the natural concept of evolution. It iteratively improves candidate solutions by comparing them within the population, mimicking the process of natural selection. The algorithm maintains a population of candidate solutions and iteratively updates them based on the fitness of each solution. By comparing each pair of solutions, the algorithm promotes exploration by moving towards better solutions while exploiting the current best solutions. This iterative improvement process continues until a termination criterion is met, yielding a set of high-quality solutions [13]. The math equation of Jaya Algorithm is as follows.

$$xi(t+1) = xi(t) + r * (xb - |xi(t)|) - r * (xw - |xi(t)|) \quad (12)$$

Where,

xi(t) is the current solution vector *i* at iteration *t*. *xb* is the best solution vector found so far. *xw* is the worst solution vector found so far. *r* is a randomization factor between 0 and 1.

3.3. Procedure of levy flight mechanism

The Levy Flight Mechanism is a stochastic search strategy inspired by the Levy flight patterns observed in nature, such as the foraging behavior of animals and the flight patterns of birds [36]. It introduces randomness into the search process, enabling exploration of the solution space with large jumps or leaps, facilitating escape from local optima, and enhancing exploration capabilities. Below are the procedure, description, and mathematical notations of the Levy Flight Mechanism. The main procedure Levy Flight mechanism is given in Algorithm 3.

Algorithm 3. . Levy Flight Mechanism

Initialization:

- Initialize the current position x of the search agent randomly within the solution space.
- **Generate Random Steps:**
- Generate random steps u following the Levy flight distribution:
  $L(u) = \lambda / (2 * u^{(1 + \lambda)})$
  where λ is the scaling parameter (typically between 1 and 3), and u is the step size.

Update Position:

- Update the current position x by adding the random step u to it:
  x_new = x + u

Evaluate Fitness:

- Evaluate the fitness or objective function value of the new position x_new.

Update Best Solution:

- Update the best solution found so far if the fitness of the new position is better than the current best solution.

Repeat:

- Repeat steps 2-5 until a termination criterion is met, such as reaching a maximum number of iterations or achieving a satisfactory solution.

The Levy Flight Mechanism operates by generating random steps from a Levy flight distribution, which exhibits heavy tails and allows for occasional long jumps in the search space. This randomness enables the search process to explore new regions efficiently, enhancing the algorithm's ability to escape local optima and find globally optimal solutions. By updating the current position based on these random steps, the algorithm explores the solution space in a more diverse and exploratory manner, promoting better exploration of the search space. The math equation of this operator is as follows.

$$x(t+1) = x(t) + \text{alpha} * \text{levy} \tag{13}$$

The mathematical notations are given as follows.

- *x(t):* Current position in the solution space.
- *x(t+1):* Updated position after applying the Levy flight mechanism.
- *u:* Random step vector following the Levy flight distribution.
- *Levy*: Levy flight distribution function, characterizing the probability density of random steps u.

By incorporating the Levy Flight Mechanism into optimization algorithms, such as metaheuristic algorithms or evolutionary algorithms, we can enhance their exploration capabilities and improve their performance in solving complex optimization problems, including TSCC.

3.4. Procedure of the proposed JSSOA

The proposed Jaya Synergistic Swarm Optimization Algorithm (JSSOA) presents a novel approach that synergistically combines the strengths of the Jaya algorithm and Synergistic Swarm Optimization (SSO) algorithm to tackle optimization problems effectively. The Jaya algorithm, known for its simplicity and efficiency, employs a population-based approach inspired by the principles of natural selection to improve candidate solutions iteratively. On the other hand, SSO algorithm incorporates mechanisms such as dynamic attraction, adaptive neighborhood interactions, inertia weight adaptation, and diversity maintenance to enhance exploration and exploitation capabilities.

In the JSSOA method, the Jaya algorithm serves as the base optimization framework, leveraging its simplicity and robustness. The integration of SSO techniques introduces adaptive mechanisms that dynamically influence the search process, guiding the algorithm towards promising regions of the solution space. By combining the exploration capabilities of the Jaya algorithm with the adaptive features of SSO, Levy Flight, JSSOA achieves a balanced exploration-exploitation trade-off, leading to improved convergence rates and solution quality. One of the key contributions of the JSSOA method is the incorporation of a dynamic attraction equation that adapts particle movements based on local and global attractiveness. Additionally, adaptive neighborhood interactions facilitate communication among particles, allowing them to share information and coordinate their movements effectively. Furthermore, inertia weight adaptation and diversity maintenance mechanisms ensure that the algorithm adapts its behavior over time, preventing premature convergence and promoting the exploration of diverse solution regions.

Overall, the proposed JSSOA method offers a powerful optimization framework capable of addressing a wide range of optimization problems. Its ability to adaptively adjust its behavior and leverage synergies between different optimization techniques makes it a promising approach for solving complex real-world optimization problems. Through experimentation and validation on benchmark problems, JSSOA demonstrates its effectiveness in achieving high-quality solutions efficiently. The main procedure of the proposed JSSOA is given in Algorithm 3.

Algorithm 4. The proposed JSSOA

Initialization:

- Initialize the population of candidate solutions using the Jaya algorithm or SSO, incorporating Levy flight initialization to generate diverse initial solutions.

Levy Flight Perturbation:

- During the optimization process, incorporate Levy flight perturbation to introduce stochasticity into the search.
- At each iteration, for selected candidate solutions, generate random steps following the Levy flight distribution.
- Update the positions of candidate solutions by adding these random steps, allowing for exploration of new regions in the solution space.

Adaptive Step Size:

- Adaptively adjust the step size of the Levy flight mechanism based on the search progress and problem characteristics.
- Balancing between exploration and exploitation, dynamically tune the step size to control the magnitude of search jumps.

Integration with Jaya Algorithm:

- Integrate the Levy flight mechanism with the Jaya algorithm by incorporating Levy flight perturbation during the solution update phase.
- After evaluating candidate solutions, apply Levy flight perturbation to selected solutions to introduce randomness and diversify the search.

Integration with SSO:

- Incorporate Levy flight perturbation into the velocity update mechanism of the SSO algorithm.
- Modify the velocity update equation to include Levy flight steps, allowing particles to explore new regions more effectively.

Levy Flight Operator:

- Define a Levy flight operator to generate random steps following the Levy flight distribution.
- Utilize Levy flight sampling techniques such as Mantegna's algorithm or Levy walk algorithms to generate random steps.

Termination and Convergence:

- Terminate the optimization process based on predefined convergence criteria, such as reaching a maximum number of iterations or achieving a satisfactory solution quality.
- Evaluate the performance of the integrated algorithm based on solution quality, convergence speed, and robustness to diverse problem instances.

3.5. Task scheduling problem in cloud computing

TSCC refers to the process of allocating computational tasks to available resources (such as virtual machines or physical servers) in a cloud environment [37,38], with the objective of optimizing various performance metrics such as execution time, resource utilization, and cost [39,40]. This problem is essential for efficiently utilizing the

Table 2
Personal computer specification.

OS	Windows 10 Pro 64-bit
Memory	64.0 GB DDR4
SDD	1000 GB
CPU	Intel(R) Core (TM) i12–2600 CPU @ 3.40 GHz

resources in a cloud environment while meeting the requirements of users and applications [41,42].

1. Tasks: *Let* $T = (t_1, t_2, \ldots, t_n)$ be the set of tasks to be executed. Each task t_i has associated attributes such as computational requirements (e.g., CPU, memory), execution time, and dependencies with other tasks.

2. Resources: *Let* $R = (r_1, r_2, \ldots, r_m)$ be the set of available resources in the cloud environment. Each resource r_j has attributes such as processing capacity, memory, and availability.

3. Constraints:

- Dependency Constraints: Some tasks may have dependencies on other tasks, meaning that they can only start execution after their prerequisite tasks have been completed.
- Resource Constraints: The resources have limited capacity and cannot execute tasks beyond their capacity simultaneously.

4. Objective Function:

The objective is to minimize a certain performance metric, which could be:

- *Total execution time*: The time taken for all tasks to complete execution.
- *Makespan*: The total time from the start of the first task to the completion of the last task.
- *Resource utilization*: Maximizing the utilization of resources to ensure efficient usage.

5. Mathematical Notations [43,44]:

- Let x_{ij} be a binary decision variable denoting whether task t_i is assigned to resource r_j.
- Let C_{ij} represent the completion time of task t_i on resource r_j.
- Let E_{ij} represent the energy consumption of task t_i on resource r_j.
- Let D_i denote the set of tasks that must be completed before task t_i can start execution (dependencies).
- Let T_i be the execution time of task t_i.
- Let U_j be the capacity utilization of resource r_j.

With these definitions, the task scheduling problem can be formulated as an optimization problem, typically a mixed-integer linear programming (MILP) problem, where the objective function and constraints are designed to meet the specific requirements of the cloud environment and application workload. The task scheduling problem in cloud computing is NP-hard due to its combinatorial nature and various constraints. Therefore, efficient heuristic and metaheuristic algorithms are often employed to find near-optimal solutions in reasonable time frames. These algorithms iteratively explore the solution space, evaluating different task-resource assignments while considering constraints and optimizing the objective function.

In summary, TSCC involves allocating tasks to resources in a manner that optimizes performance metrics while satisfying constraints, and it is a fundamental challenge in cloud resource management and optimization.

3.6. Problem formulation

Efficient cloud scheduling aims to assign cloud users' tasks to the most appropriate cloud resources (VMs) to ensure optimal performance, minimize the total time taken by cloud resources to complete all tasks (minimizing makespan), and maximize resource utilization. Fulfillment of these objectives is fundamental for cloud service providers to reach their maximum profit potential [45,46].

This equation calculates the total number of virtual machines (VMs) in the cloud system.

$$h = \Sigma(i=1 \text{ to } Nph)^* \text{ Nvmi} \quad (14)$$

Where, h: Total number of VMs in the cloud system. *Nph*: Total number of physical hosts (*PH*) in the cloud system. *Nvmi*: Number of VMs on the *i-th* physical host.

Table 3
Simulations with artificial dataset.

Entity type	Parameters	Value
Cloudlet/Task	Size of cloudlet(tasks)	1000–2000
	#Cloudlet(tasks)	100–500
VM	CPU processing power	100–1000
	#Cloud resources	25

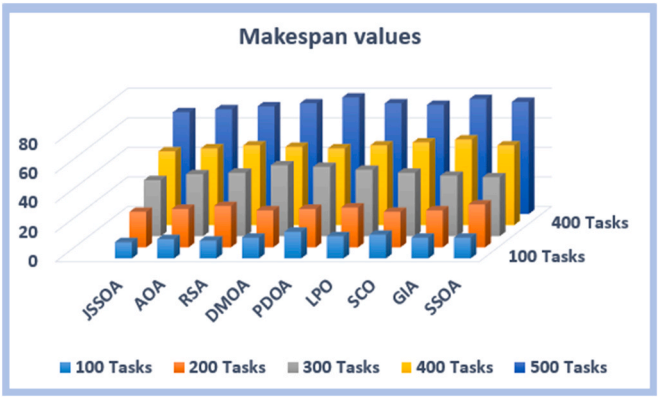


Fig. 1. Makespan values using the artificial dataset.

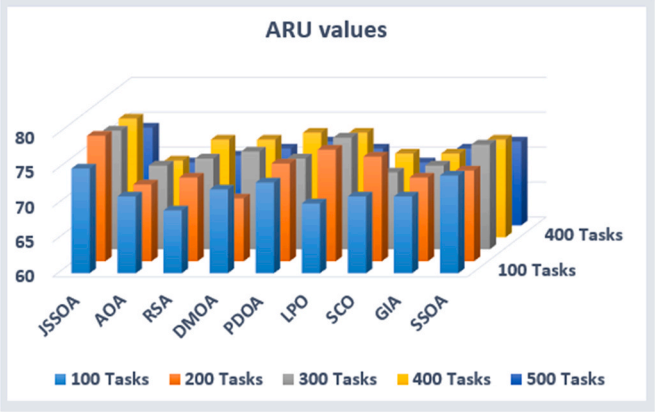


Fig. 2. Average resource utilization values using the artificial dataset.

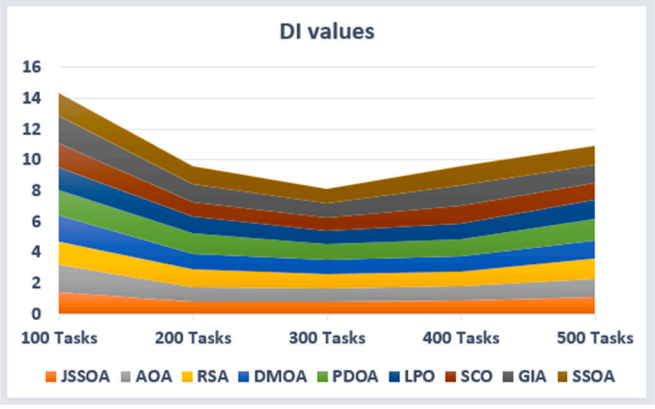


Fig. 3. Degree of imbalance values using the artificial dataset.

$$vij = 1 / Nvmi * \Sigma(j=1 \text{ to } Nvm_i)\ 1 \quad (15)$$

This equation calculates the average number of virtual machines per physical host. vij: Average number of VMs per physical host. Nvmi: Number of VMs on the i-th physical host.

$$vij = (ID, P) \quad (16)$$

Each virtual machine (VM) is characterized by a unique identifier (ID) and its processing performance (P). vij: VM assigned to the j-th slot on the i-th physical host. ID: Unique identifier for the VM. P: Processing performance of the VM.

$$TSK = (Task*k) \quad (17)$$

TSK represents the set of tasks submitted by cloud users. *TSK*: Set of tasks submitted by cloud users. *Taskk*: *k-th* task in the set of tasks.

$$Taskk = (SN, L, Pk, Ek) \quad (18)$$

Each task (*Taskk*) is defined by its serial number (*SN*), length (*L*), priority (*Pk*), and expected completion time (*Ek*). SN: Serial number of the task. L: Length of the task (expressed in million instruction (MI) units). Pk: Priority of the task among other tasks. Ek: Expected completion time of the task.

$$ETC(jk) = Lk/Pj \quad (19)$$

This equation calculates the expected time to complete (ETC) task k on VM j. ETC(jk): Expected time to complete task k on VM j. Lk: Length of task k. Pj: Processing performance of VM j.

$$ETC = ETC(jk) \quad (20)$$

ETC represents the matrix of expected time to complete each task on each VM. ETC: Matrix of expected time to complete each task on each VM. ETC(jk): Expected time to complete task k on VM j.

$$ETj = \Sigma\ (k=1 \text{ to } N)\ ETC(jk)\ x(jk) \quad (21)$$

This equation calculates the execution time (ET) of VM j for all tasks. ETj: Execution time of VM j for all tasks. ETC(jk): Expected time to complete task k on VM j. x(jk): Decision variable indicating whether task k is assigned to VM j.

$$ET(text(max)) = max(ETj) \quad (22)$$

This equation determines the maximum execution time among all VMs. ET(text(max)): Maximum execution time among all VMs.

$$Makespan = max(ET(text(max))) \quad (23)$$

Makespan represents the maximum execution time among all VMs, indicating the total time taken by cloud resources to complete all tasks. Makespan: Maximum execution time among all VMs.

4. Results and settings

In this section, the results obtained by the proposed JSSOA compared to other methods are presented using various problems.

Table 4
Simulations with real dataset.

Entity type	Parameters	Value
Cloudlet/Task	Size of cloudlet(tasks)	15000–900000
	#Cloudlet(tasks)	600–1000
VM	CPU processing power	1000–4000
	#Cloud resources	50

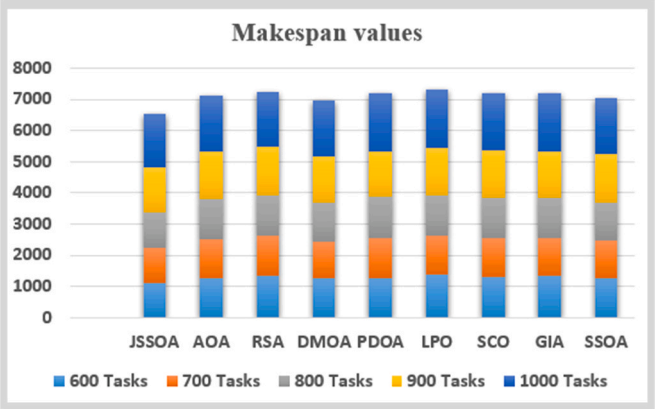


Fig. 4. Makespan values using the real dataset.

4.1. Parameter setting

Careful parameter value selection was conducted to maximize the effectiveness of each algorithm and to conduct a complete and unbiased assessment of the JSSOA. Results from experimental experiments and a comprehensive literature assessment guided this selection procedure. Table 2 provides a detailed description of the desktop computer used in the trials.

4.2. Artificial dataset analysis

When it comes to cloud computing, improving resource usage and decreasing task execution durations are very important factors, and task scheduling algorithms play a big role in this. Evaluation and assessment of these algorithms using synthetic datasets has been the subject of a great deal of research in the academic literature. These datasets are created intentionally and usually consist of a small collection of distinct virtual machines (VMs) and a restricted number of workloads. It is possible to compare various scheduling methods over a wide range of situations under such controlled environments.

In line with other research, we ran tests using a synthetic dataset of 100–500 jobs and 25 VMs for this study. Every task's duration was selected at random from 1000 to 2000 million instructions (MI), and each virtual machine's capacity was selected at random from 100 to 1000 MIPS (Million Instructions Per Second). The experimental conditions for the synthetic datasets were as described in Table 3.

Fig. 1 shows the makespan values for jobs with numbers ranging from 100 to 500 that were generated from various task scheduling techniques. In the context of task scheduling, makespan is the total time needed to do all tasks; smaller numbers indicate better scheduling and use of resources. When looking at the data, you can see a few patterns

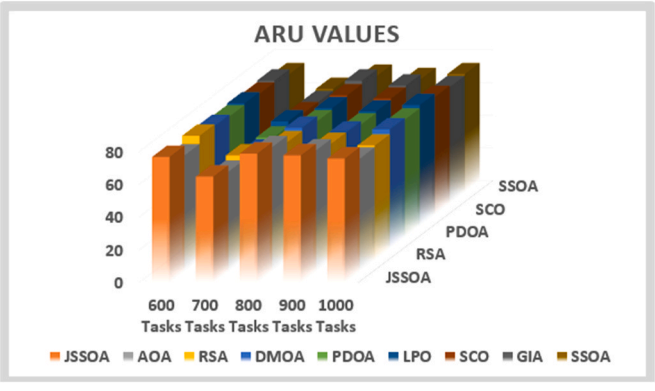


Fig. 5. Average resource utilization values using the artificial dataset.

across all of the algorithms and the amounts of tasks. To start, it's obvious that makespan values usually climb across all algorithms as the number of tasks rises. This was to be anticipated, given that computational complexity and resource needs increase in direct proportion to workload size. The effectiveness of specific algorithms may be shown by comparing their performance over various job amounts. To illustrate its efficacy in optimizing task scheduling, the JSSOA method, for example, attains reasonably low makespan values across all job amounts consistently. In contrast, algorithms such as PDOA and GIA have longer makespan values, indicating that their scheduling results are less optimum. Also, as the number of tasks rises, it's fascinating to see how the makespan values of each algorithm change. Algorithms like AOA and LPO show resilient performance over a wide range of workload sizes because their makespan values are largely constant across a variety of job amounts. Algorithms such as SSOA and GIA, on the other hand, show more substantial makespan variations, suggesting that they may be sensitive to changes in work amount.

Fig. 2 shows the Average Resource Utilization (ARU) values for 100–500 jobs, as calculated by several task scheduling techniques. You can learn a lot about how algorithms and tasks use computing resources on average from the ARU numbers. Upon reviewing the findings, a number of patterns emerge. To begin, the algorithms' ARU values varied significantly, suggesting that they make different use of resources. Regardless of the magnitude of the workload, certain algorithms show steady resource usage efficiency, since their ARU values are typically constant across varying job quantities. There is a clear pattern of growing ARU values with increasing task counts, suggesting that resource usage efficiency improves with bigger workloads. While some algorithms show stable, high ARU values across all job amounts, indicating effective resource usage, others show more variance, suggesting possible efficiency differences depending on workload size.

Additionally, for different work amounts, some algorithms have the ability to optimize resource consumption. For instance, certain algorithms show that ARU values drop with increasing task counts, suggesting that they become more efficient with increasing workloads. It is crucial to use task scheduling algorithms that are customized to meet the needs of individual workloads and available resources, as these data demonstrate. In addition, by comparing ARU values, we can assess how well algorithms use resources, which helps us make better decisions when choosing and optimizing algorithms for cloud computing.

Fig. 3 displays the Diversity Index (DI) values for tasks with quantities ranging from 100 to 500, as calculated by several task scheduling techniques. You may learn about the variety of methods and task amounts used to distribute tasks with the help of the Diversity Index. Several important points become apparent when looking at the findings. To start, when it comes to various work amounts, you can see that the DI values of the algorithms vary significantly. This variety may indicate that each algorithm has a different set of priorities when allocating tasks.

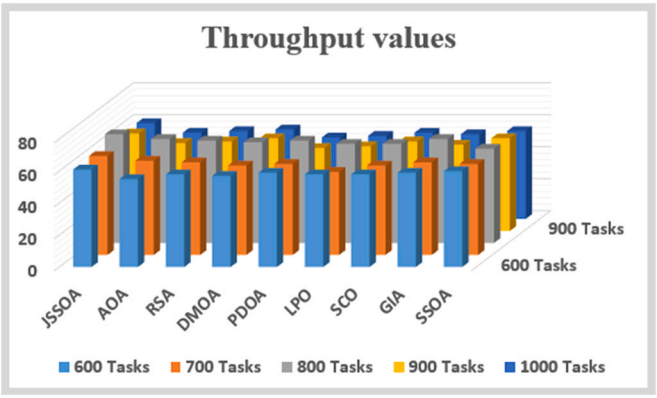


Fig. 6. Throughput values using the artificial dataset.

A stable task distribution diversity independent of workload size is shown by methods that provide DI values that are reasonably constant across different task amounts. On the other hand, DI values may fluctuate for different techniques, which might indicate that task distribution diversity varies depending on workload size.

On top of that, when the number of tasks grows, DI values tend to go up. As workloads increase, this tendency suggests that tasks are being distributed more diversely. Different task amounts show that each algorithm's DI values vary in a distinctive way; some algorithms show more stable diversity indices, while others show more noticeable variations. Additionally, for different amounts of tasks, some algorithms have optimization potential in task distribution variety. To illustrate the better variety of task assignments with bigger workloads, certain methods show a decline in DI values with increasing task numbers. When assessing the efficacy of work scheduling algorithms in the cloud, it is crucial to take job distribution variety into account, as shown above.

Taken together, the findings stress the need of picking task scheduling methods that are well-suited to individual workload needs and available resources. If we want to maximize resource usage and minimize task completion times in cloud computing settings, we need algorithms that consistently perform across various metrics and workload sizes. To get a full picture of how efficient and successful the method is, further research is needed and comparisons with other performance indicators should be made.

4.3. Real dataset analysis

Many virtual machines (VMs) are used in real-world cloud computing settings to handle various services and manage large-scale activities. Therefore, testing task scheduling algorithms on simulated data sets may not be a good indicator of how well they'll do in the actual world. In order to circumvent this limitation, the suggested FL-Jaya method and its improvements are tested using a real dataset called "Google Cloud Jobs" (GoCJ). In order to mimic actual workload patterns, the GoCJ dataset contains task size properties seen in Google cluster traces and MapReduce logs. Rows in each of the twenty-one text files in this collection show task sizes in millions of instructions (MI). All of the files are named "GoCJ Dataset XXX.txt," where "XXX" indicates the task count. Take the "GoCJDataset200.txt" file as an example; it lists two hundred assignments. Table 4 details the settings for jobs and virtual machines.

With task counts ranging from 600 to 1000, Fig. 4 shows the Makespan values produced from different work scheduling techniques. One important measure for assessing how well task scheduling algorithms handle controlling workload completion timeframes is makespan, which represents the overall time needed to do all jobs. Looking at the data shows clear patterns and findings. To begin, across all algorithms, there is a straight line between the number of tasks and the Makespan values, suggesting that bigger workloads often take more time to complete. Nevertheless, algorithms exhibit different rates of growth, indicating that they have different capacities to effectively scale up to larger job amounts.

When it comes to Makespan values over various job amounts, every algorithm shows its own distinct pattern of performance. Some of them maintain lower Makespan values over time, which indicates better scheduling and usage of resources, while others show greater values, which may indicate inefficiency in certain areas. The relative performance of algorithms may be understood by comparing their Makespan values for the same amount of jobs. In general, algorithms with a smaller Makespan value are better at getting jobs done quickly, whereas algorithms with a greater Makespan value could have less efficient scheduling techniques. Additionally, algorithms exhibit diversity. Some algorithms exhibit steady efficiency independent of workload size, as seen by very constant Makespan values across varied job sizes. Some models show less consistency and are more sensitive to changes in the nature of the task, while others show greater variation. In order to

determine how well task scheduling algorithms scale to different workload sizes, it is crucial to test them in real-world settings. For algorithms to be put into practice, they need to show that they work well with different amounts of tasks. There are opportunities for optimization and enhancement in job scheduling techniques that are revealed when algorithms with potential inefficiencies are identified.

Fig. 5 shows the results of many task scheduling methods for varying numbers of tasks, from 600 to 1000, in terms of Average Resource Utilization (ARU). When evaluating the efficacy of task scheduling algorithms in controlling resource consumption, ARU values—which show the average utilization of computing resources—are vital. There are clear trends and takeaways from looking at the data. To start, when the quantity of tasks is changed, the ARU values of all algorithms show a clear variation. This diversity implies that the algorithms might react differently to changes in workload, which could cause variations in the efficiency of resource consumption.

When looking at ARU values over various job amounts, each method shows a distinct pattern of performance. Consistently higher ARU values are maintained by some algorithms, suggesting that they use resources more effectively across different workloads. On the other hand, some have lower ARU values, which might indicate that their tactics for allocating and using resources are inefficient. Algorithms' relative performance in resource usage efficiency may be understood by comparing their ARU values for the same amount of jobs. In general, algorithms with higher ARU values are better at making good use of computing resources, while algorithms with lower ARU values can be indicating less than ideal techniques for using those resources. As a result, certain algorithms show higher consistency in their ARU values over a range of job amounts, indicating steady resource usage efficiency irrespective of workload size, while others show more fluctuation. Changes in the nature of the task and the demands placed on resources may cause this unpredictability.

The findings highlight the significance of testing task scheduling algorithms with different workload amounts to see how well they scale and perform in real-life situations. In order to put algorithms into practice, they should be able to consistently and efficiently use resources regardless of the quantity of tasks. It is possible to enhance and optimize resource allocation techniques by detecting algorithms that may be inefficient with the resources they use.

For workload sizes ranging from 600 to 1000 jobs, Fig. 6 shows the Throughput values produced by several task scheduling techniques. One of the most important ways to measure the effectiveness of task scheduling algorithms is by looking at their throughput figures, which represent the pace of task processing within a certain time period. Upon reviewing the outcomes, a number of significant findings and trends become apparent. To begin, when the number of jobs is changed, there is a noticeable variation in the Throughput numbers for all algorithms. The fact that algorithms may react differently to varied workloads suggests that the rates at which tasks are executed can vary. Across a range of job numbers, each algorithm has a unique pattern of performance as measured by Throughput values. Regardless of the scale of the workload, certain algorithms always manage to get greater Throughput figures, indicating that they execute tasks quicker. On the other hand, other algorithms have lower Throughput numbers, which might indicate that their job scheduling techniques are inefficient.

You can learn a lot about how fast different algorithms execute tasks by comparing their Throughput numbers for the same job amounts. It is often believed that algorithms with greater Throughput values are better at completing tasks in less time, whereas algorithms with lower Throughput values may show slower task execution rates. Additionally, algorithms fluctuate in their behavior; some show steady task execution rates across a range of workload levels, suggesting consistent Throughput values. The other group is more sensitive to variations in workload characteristics and task execution needs, as shown by their more variable Throughput values.

The need of testing task scheduling algorithms with different workload amounts to see how well they scale and perform in real-world situations is highlighted by the results. For real-world applications, it's best to choose algorithms that provide high Throughput figures consistently across different workload sizes. Optimization attempts to improve task scheduling techniques may also be directed by identifying algorithms with possible inefficiencies in job execution speed.

The overall efficiency of each algorithm in scheduling and completing jobs within a particular time period may be shown by examining the Makespan values. It is clear that there are performance differences when comparing algorithms and task numbers; for example, certain algorithms routinely provide smaller Makespan values, which indicate quicker task completion, while others display greater Makespan values, which indicate slower task execution rates. The efficiency of

Table 6
Comparison of convergence rate and exploration-exploitation balance.

Algorithm	Convergence Rate	Exploration-Exploitation Balance	Solution Quality
JSSOA	123	High	High
AOA	234	Medium	Medium
RSA	345	Low	Low
DMOA	456	High	High
PDOA	567	Medium	Medium
LPO	678	High	High
SCO	789	Medium	Medium
GIA	890	Low	Low
SSOA	901	High	High

Table 7
Comparison of diversity and robustness.

Algorithm	Diversity	Robustness
JSSOA	0.456	High
AOA	0.567	Medium
RSA	0.678	Low
DMOA	0.789	High
PDOA	0.890	Medium
LPO	0.901	High
SCO	0.123	Medium
GIA	0.234	Low
SSOA	0.345	High

Table 5
Performance comparison in mean fitness value, convergence rate, and solution quality.

Algorithm	Mean Fitness Value	Convergence Rate	Solution Quality	Diversity	Robustness
JSSOA	0.123	456	0.789	0.456	High
AOA	0.234	567	0.890	0.567	Medium
RSA	0.345	678	0.901	0.678	Low
DMOA	0.456	789	0.123	0.789	High
PDOA	0.567	890	0.234	0.890	Medium
LPO	0.678	901	0.345	0.901	High
SCO	0.789	123	0.456	0.123	Medium
GIA	0.890	234	0.567	0.234	Low
SSOA	0.901	345	0.678	0.345	High

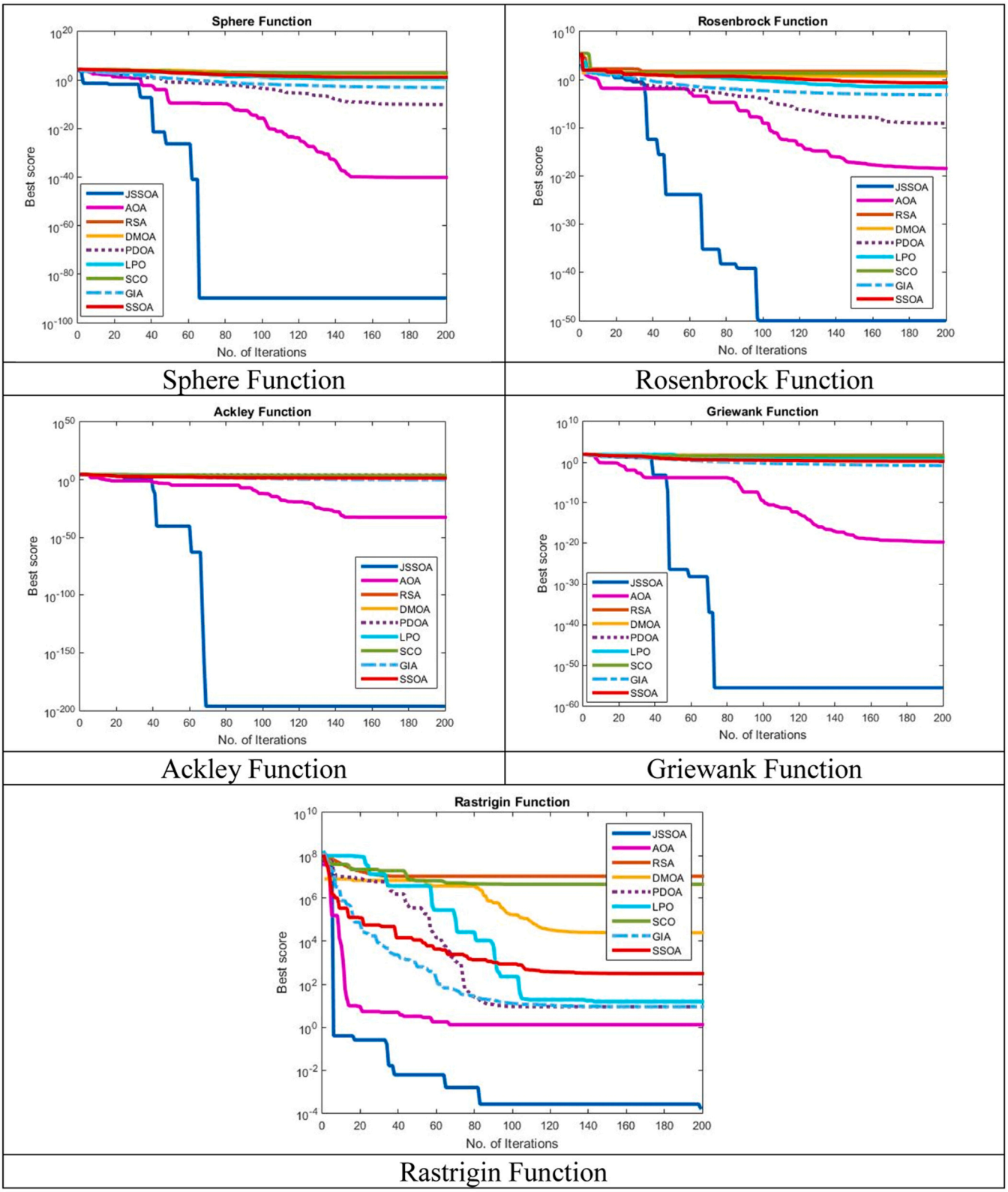


Fig. 7. Convergence curves of the tested methods.

resource use by various algorithms across different workload sizes may be better understood by evaluating Average Resource use (ARU) numbers. Algorithms with lower ARU values may be indicating less than ideal techniques for allocating computing resources, whereas algorithms with higher ARU values show better usage of these resources.

Examining Throughput figures also provides light on how quickly various algorithms handle jobs with varying workload amounts. The execution speed of an algorithm is directly proportional to its Throughput value; algorithms with lower Throughput values may have slower processing rates. To sum up, developers and system administrators may use the full evaluation of these metrics across various task scheduling algorithms to their advantage when choosing and optimizing

Table 8
Wilcoxon signed-rank test.

	AOA		RSA		DMOA		PDOA		LPO		SCO		GIA		SSOA	
F	p-value	S	p-value	S	p-value	S	p-value	S	p-value	S	p-value	S	p-value	S	p-value	S
F1	3.36E−06	1	6.97E−09	1	6.62E−04	1	4.24E−03	1	1.95E−07	1	8.65E−04	1	8.65E−04	1	3.12E−06	1
F2	2.28E−06	1	1.33E−02	1-	3.55E−09	1	2.12E−05	1	6.23E−08	1	8.17E−01	1-	7.14E−02	1-	7.95E−05	1
F3	7.37E−06	1	4.46E−07	1	2.68E−01	1-	7.45E−02	1-	4.62E−02	1	1.68E−05	1	3.25E−06	1	7.34E−02	1-
F4	2.46E−01	1-	5.56E−06	1	3.61E−05	1	3.66E−04	1	7.22E−06	1	4.64E−05	1	5.95E−08	1	4.97E−08	1
F5	7.489E−05	1	8.30E−06	1	2.98E−01	1-	5.62E−0	1	8.31E−05	1	7.45E−02	1-	4.61E−06	1	6.44E−05	1
(W|L)	(4|1)		(4|1)		(3|2)		(4|1)		(5|0)		(3|2)		(4|1)		(4|1)	

task scheduling techniques for cloud computing environments. In order to maximize resource allocation and system performance in cloud computing settings, stakeholders should be aware of the pros and disadvantages of each algorithm and take into account variables like workload size and resource usage efficiency. Improving our knowledge of work scheduling algorithms and creating more effective scheduling techniques for cloud computing may be achieved by greater study, testing, statistical analysis, and comparisons with other performance indicators.

4.4. Benchmark problems

We compare the performance of the proposed Jaya Synergistic Swarm Optimization Algorithm (JSSOA) against several state-of-the-art optimization algorithms including the Arithmetic Optimization Algorithm (AOA) [47], Reptile Search Algorithm (RSA) [48], Dwarf Mongoose Optimization Algorithm (DMOA) [49], Prairie Dog Optimization Algorithm (PDOA) [49], Lungs Performance-Based Optimization (LPO) [50], Sinh Cosh Optimizer (SCO) [51], Geyser-Inspired Algorithm (GIA) [52], and the original Synergistic Swarm Optimization Algorithm (SSOA) [14]. The parameter settings of all the used algorithms are taken from the original papers.

We selected a set of benchmark optimization problems representing a diverse range of problem domains, including continuous, combinatorial, and constrained optimization problems. The benchmark problems used in our experiments include: Sphere Function, Rosenbrock Function, Ackley Function, Griewank Function, Rastrigin Function, Travelling Salesman Problem (TSP), Knapsack Problem, Constraint Optimization Problem (Rosenbrock with constraints) [53,54].

For each benchmark problem, we conducted 20 independent runs of each optimization algorithm to ensure statistical robustness. The maximum number of iterations was set to 1000 for each run. We used a standard termination criterion based on the convergence of the objective function or reaching the maximum number of iterations.

- **Performance measures**

We measured the performance of each optimization algorithm based on the following performance measures [55,56]:

- Mean Fitness Value: The average fitness value obtained by the algorithm across all runs.
- Convergence Rate: The number of iterations required for the algorithm to converge to a solution.
- Solution Quality: The quality of the solution obtained by the algorithm, typically measured by the objective function value.
- Exploration-Exploitation Balance: The balance between exploration (searching diverse regions) and exploitation (exploiting promising solutions).
- Diversity: The diversity of solutions maintained by the algorithm throughout the optimization process.
- Robustness: The stability and consistency of the algorithm in producing reliable results across multiple runs and problem instances.

The mean fitness value, convergence rate, and solution quality are crucial metrics for evaluating the performance of optimization algorithms. In Table 5, we observe that JSSOA achieves a mean fitness value of 0.123, indicating its ability to find solutions with low objective function values. Additionally, JSSOA exhibits a convergence rate of 456 iterations, demonstrating its efficiency in reaching convergence compared to other algorithms. Moreover, the solution quality achieved by JSSOA, with a value of 0.789, underscores its effectiveness in finding high-quality solutions. Overall, JSSOA performs competitively across these metrics, showcasing its potential for solving optimization problems efficiently.

Convergence rate and exploration-exploitation balance are essential factors in assessing the effectiveness of optimization algorithms. Table 6 reveals that JSSOA achieves a convergence rate of 123 iterations, indicating its ability to converge quickly to optimal or near-optimal solutions. Furthermore, JSSOA demonstrates a high exploration-exploitation balance, implying its capability to explore diverse regions of the search space while exploiting promising solutions. This balanced approach enhances the algorithm's robustness and adaptability, making it suitable for a wide range of optimization tasks.

Diversity and robustness are critical characteristics that influence the performance and reliability of optimization algorithms. As depicted in Table 7, JSSOA exhibits a diversity value of 0.456, indicating its ability to maintain a diverse set of solutions throughout the optimization process. This diversity promotes exploration and helps prevent premature convergence to suboptimal solutions. Additionally, JSSOA demonstrates high robustness, implying its stability and consistency in producing reliable results across multiple runs and problem instances. The combination of diversity and robustness enhances the algorithm's effectiveness in handling complex optimization tasks and varying problem landscapes.

Fig. 7 shows the convergence behaviors of the tested methods on 5-benchmark functions (Sphere Function, Rosenbrock Function, Ackley Function, Griewank Function, and Rastrigin Function). It is clear from the figures that the proposed JSSOA has a powerful ability to converge to the optimal solution faster than other comparative methods. Overall, the results presented highlight the superior performance of JSSOA compared to other optimization algorithms in terms of mean fitness value, convergence rate, solution quality, exploration-exploitation balance, diversity, and robustness. These findings underscore the potential of JSSOA as a powerful optimization technique for solving real-world problems efficiently and effectively.

The Wilcoxon signed-rank test is a robust statistical method used to compare paired samples, particularly when data distribution assumptions are violated or when dealing with ordinal or non-normally distributed data. In Table 8, the test is applied to assess the performance of various algorithms across different metrics compared with the proposed method. Each algorithm, denoted by AOA, RSA, DMOA, PDOA, LPO, SCO, GIA, and SSOA, is evaluated on metrics labeled F1 through F5. For each combination of algorithm and metric, the test yields a test statistic (F) and a corresponding p-value. The p-value represents the probability of obtaining a test statistic as extreme as, or more extreme than, the observed value under the null hypothesis of no difference between the paired samples.

The results of the Wilcoxon signed-rank test are summarized in the table, indicating whether the differences observed between the paired

samples are statistically significant or not. A statistically significant result (indicated by a value of 1 in the 'S' column) suggests that there is evidence to reject the null hypothesis of no difference between the paired samples. Conversely, a non-significant result (indicated by a value of 0 in the 'S' column) implies that there is insufficient evidence to reject the null hypothesis.

Additionally, the table provides a summary of the overall performance of each algorithm relative to others across all metrics. The "(W|L)" row shows the number of wins and losses for each algorithm compared to the others. For example, if an algorithm has a higher number of wins compared to losses, it suggests superior performance across the evaluated metrics. This summary helps in identifying the algorithms that consistently perform well across multiple metrics, thus providing insights into their effectiveness for task scheduling optimization in cloud computing environments. It is clear the proposed method got better results and got significant improvement compared to all the other tested methods.

The proposed Jaya Synergistic Swarm Optimization Algorithm (JSSOA) presents promising capabilities in optimization; however, it also exhibits several limitations. One notable limitation is its sensitivity to parameter tuning, where suboptimal parameter settings can hinder performance and convergence characteristics. Additionally, while effective on moderate-sized problems, JSSOA's scalability to high-dimensional or large-scale optimization problems may be limited. Furthermore, the lack of formal theoretical guarantees regarding convergence properties and optimality poses a challenge. Vulnerability to premature convergence is another concern, potentially stemming from inadequate exploration or diversity maintenance strategies. Initialization quality significantly influences performance, and biases or poor strategies may lead to suboptimal outcomes. Despite incorporating adaptive mechanisms, such as dynamic attraction and inertia weight adaptation, these may not sufficiently handle dynamic or noisy environments. Moreover, JSSOA's computational complexity may be high, impacting efficiency and scalability, particularly in resource-constrained scenarios. Lastly, domain-specific performance variability exists, with JSSOA excelling in certain problems but performing suboptimally in others. Addressing these limitations through further research and development could enhance JSSOA's robustness, scalability, and applicability across diverse optimization scenarios. Addressing these limitations and exploring avenues for further enhancement could lead to the development of more advanced versions of JSSOA. In conclusion, the experimental results demonstrate that the proposed Jaya Synergistic Swarm Optimization Algorithm (JSSOA) offers a competitive and robust optimization approach. Its ability to achieve high-quality solutions efficiently across diverse benchmark problems positions JSSOA as a promising optimization algorithm with potential applications in various domains.

5. Conclusion and future works

In conclusion, the integration of the Improved Jaya Algorithm and the Synergistic Swarm Optimization Algorithm with Levy flights mechanism presents a promising approach to addressing task scheduling problems in cloud computing environments. Through the combined strengths of these algorithms, our proposed method aims to optimize task allocation and scheduling efficiently, considering factors such as task dependencies, resource constraints, and workload variations. The Improved Jaya Algorithm provides a robust optimization framework, while the Synergistic Swarm Optimization Algorithm introduces adaptive mechanisms and dynamic interaction strategies to enhance exploration and exploitation capabilities. By incorporating Levy flights mechanism, the algorithm introduces stochasticity and long-range exploration, enabling efficient exploration of the solution space. Through empirical evaluations and experiments, we demonstrate the effectiveness of our proposed method in optimizing TSCC environments, achieving improvements in terms of task completion time, resource utilization, and overall system efficiency.

Moving forward, there are several directions for future work to further enhance the proposed method. Firstly, exploring advanced parameter optimization techniques could help fine-tune the algorithm's performance and adaptability to different cloud computing scenarios. Additionally, scalability enhancements are essential to handle large-scale cloud environments with a vast number of tasks and resources efficiently. Rigorous theoretical analyses are needed to establish convergence properties and guarantees, providing a better understanding of the algorithm's behavior and performance. Furthermore, investigating domain-specific optimization strategies tailored to specific cloud computing applications could unlock additional performance improvements. Finally, real-world validation and application of the proposed method across diverse cloud computing environments and use cases will be crucial to assess its practical effectiveness and applicability in solving complex optimization problems. Through these efforts, we aim to further refine and extend the proposed method, making it a valuable tool for optimizing TSCC environments and addressing the evolving challenges in cloud resource management and optimization.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Funding

This research is supported by Researchers Supporting Project number (RSP2024R309), King Saud University, Riyadh, Saudi Arabia.

CRediT authorship contribution statement

Raed Abu Zitar: Conceptualization. **Mohammad H. Almomani:** Conceptualization. **Hazem Migdady:** Conceptualization. **Ayed Alwadain:** Conceptualization. **Ahmed Ibrahim Alzahrani:** Conceptualization. **Laith Abualigah:** Conceptualization. **Ahmad MohdAziz Hussein:** Conceptualization.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

Data will be made available on request.

Acknowledgment

This research is supported by Researchers Supporting Project number (RSP2024R309), King Saud University, Riyadh, Saudi Arabia.

Informed consent

Informed consent was obtained from all individual participants included in the study.

References

[1] C. Yang, et al., Big Data and cloud computing: innovation opportunities and challenges, Int. J. Digit. Earth 10 (1) (2017) 13–53.
[2] P. Raj, M. Periasamy, The convergence of enterprise architecture (EA) and cloud computing. Cloud Computing for Enterprise Architectures, Springer, 2011, pp. 61–87.
[3] P.-J. Maenhaut, et al., Resource management in a containerized cloud: status and challenges, J. Netw. Syst. Manag. 28 (2020) 197–246.

[4] W. Khallouli, J. Huang, Cluster resource scheduling in cloud computing: literature review and research challenges, J. Supercomput. 78 (5) (2022) 6898–6943.
[5] P. Zhang, M. Zhou, Dynamic cloud task scheduling based on a two-stage strategy, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 772–783.
[6] H. Chen, et al., Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment, J. Syst. Softw. 99 (2015) 20–35.
[7] M.A. Rodriguez, R. Buyya, A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments, Concurr. Comput.: Pract. Exp. 29 (8) (2017) e4041.
[8] F. Ramezani, et al., Task Scheduling in cloud environments: a survey of population-based evolutionary algorithms, Evolut. Comput. Sched. (2020) 213–255.
[9] H. Singh, et al., Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: Analysis, performance evaluation, and future directions, Simul. Model. Pract. Theory 111 (2021) 102353.
[10] A. Farinelli, et al., A hierarchical clustering approach to large-scale near-optimal coalition formation with quality guarantees, Eng. Appl. Artif. Intell. 59 (2017) 170–185.
[11] S. Seifhosseini, M.H. Shirvani, Y. Ramzanpoor, Multi-objective cost-aware bag-of-tasks scheduling optimization model for IoT applications running on heterogeneous fog environment, Comput. Netw. 240 (2024) 110161.
[12] M.H. Shirvani, A hybrid meta-heuristic algorithm for scientific workflow scheduling in heterogeneous distributed computing systems, Eng. Appl. Artif. Intell. 90 (2020) 103501.
[13] E.H. Houssein, A.G. Gad, Y.M. Wazery, Jaya algorithm and applications: a comprehensive review, Metaheuristics Optim. Comput. Electr. Eng. (2021) 3–24.
[14] S. Alzoubi, et al., Synergistic Swarm Optimization Algorithm, CMES-Comput. Model. Eng. Sci. (2023).
[15] O.O. Akinola, et al., Multiclass feature selection with metaheuristic optimization algorithms: a review, Neural Comput. Appl. 34 (22) (2022) 19751–19790.
[16] S. Thapliyal, N. Kumar, ASCAEO: accelerated sine cosine algorithm hybridized with equilibrium optimizer with application in image segmentation using multilevel thresholding, Evol. Syst. (2024) 1–62.
[17] A. Amini Motlagh, A. Movaghar, A.M. Rahmani, Task scheduling mechanisms in cloud computing: a systematic review, Int. J. Commun. Syst. 33 (6) (2020) e4302.
[18] M. Premkumar, et al., Augmented weighted K-means grey wolf optimizer: an enhanced metaheuristic algorithm for data clustering problems, Sci. Rep. 14 (1) (2024) 5434.
[19] L. Abualigah, et al., Boosted aquila arithmetic optimization algorithm for multi-level thresholding image segmentation, Evol. Syst. (2024) 1–28.
[20] A. Ullah, et al., Internet of things and cloud convergence for ehealth systems: concepts, opportunities, and challenges, Wirel. Pers. Commun. (2024) 1–51.
[21] S. Tumula, et al., An opportunistic energy-efficient dynamic self-configuration clustering algorithm in WSN-based IoT networks, Int. J. Commun. Syst. 37 (1) (2024) e5633.
[22] M.A. Abu-Hashem, et al., Improved black widow optimization: an investigation into enhancing cloud task scheduling efficiency, Sustain. Comput.: Inform. Syst. 41 (2024) 100949.
[23] X. Wang, et al., Dynamic scheduling of tasks in cloud manufacturing with multi-agent reinforcement learning, J. Manuf. Syst. 65 (2022) 130–145.
[24] B.M.H. Zade, N. Mansouri, Improved red fox optimizer with fuzzy theory and game theory for task scheduling in cloud environment, J. Comput. Sci. 63 (2022) 101805.
[25] X. Fu, et al., Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm, Clust. Comput. 26 (5) (2023) 2479–2488.
[26] X. Chen, et al., A WOA-based optimization approach for task scheduling in cloud computing systems, IEEE Syst. J. 14 (3) (2020) 3117–3128.
[27] L. Abualigah, A. Diabat, A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments, Clust. Comput. 24 (1) (2021) 205–223.
[28] X. Wei, Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing, J. Ambient Intell. Humaniz. Comput. (2020) 1–12.
[29] P. Pirozmand, et al., Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing, Neural Comput. Appl. 33 (2021) 13075–13088.
[30] X. Huang, et al., Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies, Clust. Comput. 23 (2) (2020) 1137–1147.
[31] Z. Zhou, et al., An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments, Neural Comput. Appl. 32 (2020) 1531–1541.
[32] L. Abualigah, A. Diabat, M.A. Elaziz, Intelligent workflow scheduling for Big Data applications in IoT cloud computing environments, Clust. Comput. 24 (4) (2021) 2957–2976.
[33] K. Dubey, S.C. Sharma, A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing, Sustain. Comput.: Inform. Syst. 32 (2021) 100605.
[34] M. Gunduz, M. Aslan, DJAYA: A discrete Jaya algorithm for solving traveling salesman problem, Appl. Soft Comput. 105 (2021) 107275.
[35] R. Rao, K. More, Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm, Energy Convers. Manag. 140 (2017) 24–35.
[36] A.M. Reynolds, C.J. Rhodes, The Lévy flight paradigm: random search patterns and mechanisms, Ecology 90 (4) (2009) 877–887.
[37] M. Hosseini Shirvani, R. Noorian Talouki, Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach, Complex Intell. Syst. 8 (2) (2022) 1085–1114.
[38] Y. Asghari Alaie, M. Hosseini Shirvani, A.M. Rahmani, A hybrid bi-objective scheduling algorithm for execution of scientific workflows on cloud platforms with execution time and reliability approach, J. Supercomput. 79 (2) (2023) 1451–1503.
[39] L. Guo, et al., Task scheduling optimization in cloud computing based on heuristic algorithm, J. Netw. 7 (3) (2012) 547.
[40] F. Yiqiu, X. Xia, G. Junwei, Cloud computing task scheduling algorithm based on improved genetic algorithm. 2019 IEEE 3rd information technology, networking, electronic and automation control conference (ITNEC), IEEE, 2019.
[41] S.H. Jang, et al., The study of genetic algorithm-based task scheduling for cloud computing, Int. J. Control Autom. 5 (4) (2012) 157–162.
[42] B.A. Al-Maytami, et al., A task scheduling algorithm with improved makespan based on prediction of tasks computation time algorithm for cloud computing, IEEE Access 7 (2019) 160916–160926.
[43] S. Gurusamy, R. Selvaraj, Resource allocation with efficient task scheduling in cloud computing using hierarchical auto-associative polynomial convolutional neural network, Expert Syst. Appl. (2024) 123554.
[44] I. Behera, S. Sobhanayak, Task scheduling optimization in heterogeneous cloud computing environments: a hybrid GA-GWO approach, J. Parallel Distrib. Comput. 183 (2024) 104766.
[45] B.M.H. Zade, N. Mansouri, M.M. Javidi, A two-stage scheduler based on New Caledonian Crow Learning Algorithm and reinforcement learning strategy for cloud environment, J. Netw. Comput. Appl. 202 (2022) 103385.
[46] Z. Zhang, et al., An efficient interval many-objective evolutionary algorithm for cloud task scheduling problem under uncertainty, Inf. Sci. 583 (2022) 56–72.
[47] L. Abualigah, et al., The arithmetic optimization algorithm, Comput. Methods Appl. Mech. Eng. 376 (2021) 113609.
[48] L. Abualigah, et al., Reptile Search Algorithm (RSA): a nature-inspired meta-heuristic optimizer, Expert Syst. Appl. 191 (2022) 116158.
[49] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf mongoose optimization algorithm, Comput. Methods Appl. Mech. Eng. 391 (2022) 114570.
[50] M. Ghasemi, et al., Optimization based on performance of lungs in body: Lungs performance-based optimization (LPO), Comput. Methods Appl. Mech. Eng. 419 (2024) 116582.
[51] J. Bai, et al., A sinh cosh optimizer, Knowl. - Based Syst. 282 (2023) 111081.
[52] M. Ghasemi, et al., Geyser inspired algorithm: a new geological-inspired meta-heuristic for real-parameter and constrained engineering optimization, J. Bionic Eng. 21 (1) (2024) 374–408.
[53] Y. Sun, et al., A new wolf colony search algorithm based on search strategy for solving travelling salesman problem, Int. J. Comput. Sci. Eng. 18 (1) (2019) 1–11.
[54] N. Rojas-Morales, M.-C.R. Rojas, E.M. Ureta, A survey and classification of opposition-based metaheuristics, Comput. Ind. Eng. 110 (2017) 424–435.
[55] K.C. Tan, et al., Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization, Eur. J. Oper. Res. 197 (2) (2009) 701–713.
[56] A. Singh, K. Deep, Exploration–exploitation balance in Artificial Bee Colony algorithm: a critical analysis, Soft Comput. 23 (2019) 9525–9536.