Received: 22 December 2023 | Revised: 25 April 2024 | Accepted: 11 June 2024

DOI: 10.1002/ett.5019

RESEARCH ARTICLE

WILEY

GIJA:Enhanced geyser-inspired Jaya algorithm for task scheduling optimization in cloud computing

Laith Abualigah[1,2,3] | **Ahmad MohdAziz Hussein**[4] | **Mohammad H. Almomani**[5] | **Raed Abu Zitar**[6] | **Mohammad Sh. Daoud**[7] | **Hazem Migdady**[8] | **Ahmed Ibrahim Alzahrani**[9] | **Ayed Alwadain**[9]

[1]Computer Science Department, Al al-Bayt University, Mafraq, Jordan

[2]Jadara Research Center, Jadara University, Irbid, Jordan

[3]Applied Science Research Center, Applied Science Private University, Amman, Jordan

[4]Department of Computer Science, Faculty of Information Technology, Middle East University, Amman, Jordan

[5]Department of Mathematics, Facility of Science, The Hashemite University, Zarqa, Jordan

[6]Sorbonne Center of Artificial Intelligence, Sorbonne University, Paris, France

[7]College of Engineering, Al Ain University, Abu Dhabi, United Arab Emirates

[8]CSMIS Department, Oman College of Management and Technology, Barka, Oman

[9]Computer Science Department, Community College, King Saud University, Riyadh, Saudi Arabia

Correspondence
Laith Abualigah, Computer Science Department, Al al-Bayt University, Mafraq 25113, Jordan.
Email: aligah.2020@gmail.com

Funding information
This work is funded by the Researchers Supporting Project number (RSP2024R157), King Saud University, Riyadh, Saudi Arabia

Abstract

Task scheduling optimization plays a pivotal role in enhancing the efficiency and performance of cloud computing systems. In this article, we introduce GIJA (Geyser-inspired Jaya Algorithm), a novel optimization approach tailored for task scheduling in cloud computing environments. GIJA integrates the principles of the Geyser-inspired algorithm with the Jaya algorithm, augmented by a Levy Flight mechanism, to address the complexities of task scheduling optimization. The motivation for this research stems from the increasing demand for efficient resource utilization and task management in cloud computing, driven by the proliferation of Internet of Things (IoT) devices and the growing reliance on cloud-based services. Traditional task scheduling algorithms often face challenges in handling dynamic workloads, heterogeneous resources, and varying performance objectives, necessitating innovative optimization techniques. GIJA leverages the eruptive dynamics of geysers, inspired by nature's efficiency in channeling resources, to guide task scheduling decisions. By combining this Geyser-inspired approach with the simplicity and effectiveness of the Jaya algorithm, GIJA offers a robust optimization framework capable of adapting to diverse cloud computing environments. Additionally, the integration of the Levy Flight mechanism introduces stochasticity into the optimization process, enabling the exploration of solution spaces and accelerating convergence. To evaluate the efficacy of GIJA, extensive experiments are conducted using synthetic and real-world datasets representative of cloud computing workloads. Comparative analyses against existing task scheduling algorithms, including AOA, RSA, DMOA, PDOA, LPO, SCO, GIA, and GIAA, demonstrate the superior performance of GIJA in terms of solution quality, convergence rate, diversity, and robustness. The findings of GIJA provide a promising solution quality for addressing the complexities of task scheduling in cloud environments (95%), with implications for enhancing system performance, scalability, and resource utilization.

Abbreviations: $f(X_i)$, fitness function; p_i, the probability of selecting a member from the population; N_c, the population size; X_c, alternative path; X, solution; d, the distance value; $d(X,Y)$, the distance between members of X and Y; D, decision variables; X_n, neighboring solution; Rand, random number; Iter, iteration number; MIter, max number of iterations; $X(t)$, the current solution; x_w, the least favorable solution vector; r, a randomization factor; u, the probability density of a series of random steps; T, a set of task; R, the available resources; r, resource; t_n, max number of tasks; $C\mathbb{T}_j$, the duration for completing task t on resource r; $E\mathbb{T}$, the energy consumption; $U\mathbb{T}$, the capacity utilization of resource r; h, the total number of VMs; N_{ph}, the total number of physical hosts; v_{ij}, the typical number of VMs hosted; N_{vmi}, the number of VMs running; ETC(jk), the anticipated time to complete task k on VM j; L_k, the length of task k; ET_j, the time taken by VM j to complete all tasks; P_j, the processing performance of VM j; JCM, job completion time.

1 | INTRODUCTION

In recent years, the proliferation of Internet of Things (IoT) devices and the exponential growth of data have led to an unprecedented demand for computational resources.[1,2] Cloud computing has emerged as a pivotal technology to meet these demands by providing scalable and on-demand access to computing resources over the internet.[3,4] However, efficient utilization of these resources remains a significant challenge, particularly in the context of task scheduling optimization.[5,6]

In today's era of digital transformation, where data are increasing at an unprecedented rate and computing needs are soaring, cloud computing has emerged as a pivotal technology.[7,8] Offering scalable and on-demand access to computing resources over the internet, cloud computing has revolutionized the way businesses and individuals utilize and manage computational resources.[9,10] However, one of the critical challenges in cloud computing remains the efficient allocation and scheduling of tasks across available resources.[11–13]

Task scheduling lies at the heart of optimizing resource utilization and performance in cloud computing environments.[14,15] It involves allocating computational tasks to appropriate resources to minimize execution time, reduce energy consumption, and maximize system throughput.[16,17] However, the task scheduling problem in cloud computing is complex and multifaceted, characterized by dynamic workloads, fluctuating resource availability, and diverse application requirements.[18,19]

Cloud computing environments host a myriad of applications and services, ranging from web hosting and e-commerce platforms to big data analytics and machine learning algorithms.[20–22] Each of these applications has unique computational requirements and performance objectives, further complicating the task scheduling process.[23,24] Moreover, with the advent of IoT devices and edge computing, the task scheduling problem becomes even more challenging, as tasks need to be efficiently distributed across edge devices and cloud servers.[25,26] Traditional task scheduling algorithms often fall short in addressing the complexities of cloud computing environments. They may struggle to adapt to dynamic workloads, leading to suboptimal resource utilization and performance degradation.[27] As a result, there is a growing need for innovative optimization techniques and algorithms that can effectively address the task scheduling problem in cloud computing.

To address these challenges, researchers have explored various optimization techniques and algorithms. One promising approach is the integration of nature-inspired algorithms, which mimic natural phenomena to solve optimization problems efficiently. In this context, the GIJA emerges as a novel solution for task scheduling optimization in cloud computing environments. GIJA combines the principles of the Jaya algorithm and the Geyser-inspired algorithm, augmented by the Levy Flight search mechanism. The Jaya algorithm, known for its simplicity and effectiveness, iteratively improves candidate solutions by emulating the principles of natural selection. Meanwhile, the Geyser-inspired algorithm draws inspiration from the eruptive behavior of geysers, leveraging dynamic attraction mechanisms to explore solution spaces efficiently. The integration of the Levy Flight search mechanism further enhances the algorithm's exploration capabilities by allowing for large jumps or leaps in the search space. The main contributions of this work are given as follows.

1. This paper introduces the GIJA, a novel hybrid algorithm that combines the Jaya algorithm, the Geyser-Inspired algorithm, and the Levy Flight search mechanism for task scheduling optimization in cloud computing environments.
2. GIJA offers improved performance compared to traditional task scheduling algorithms by leveraging nature-inspired principles to explore solution spaces and find optimal or near-optimal solutions efficiently.
3. By optimizing task scheduling, GIJA enhances resource utilization in cloud computing environments, leading to reduced execution times, lower energy consumption, and increased system throughput.
4. GIJA is designed to be adaptable and scalable, capable of addressing diverse application requirements and dynamically changing environments in cloud computing.
5. The proposed GIJA algorithm has practical applications in various domains, including IoT, big data analytics, and distributed computing, where efficient resource management is critical for performance and cost optimization.

This article addresses the pressing challenge of task scheduling optimization in cloud computing environments amidst the rapid proliferation of IoT devices and exponential data growth. Despite the pivotal role of cloud computing in meeting escalating computational demands, efficient resource allocation remains a significant hurdle. Task scheduling optimization, crucial for enhancing resource utilization and performance, is particularly complex due to dynamic workloads, fluctuating resource availability, and diverse application requirements. Traditional algorithms often fall short in adapting

to these complexities, necessitating innovative approaches. This study introduces the GIJA, a hybrid solution amalgamating the Jaya algorithm, Geyser-inspired algorithm, and Levy Flight search mechanism. GIJA stands out for its ability to efficiently explore solution spaces, leading to improved task scheduling performance in cloud computing environments. By enhancing resource utilization, GIJA reduces execution times, lowers energy consumption, and boosts system throughput. Moreover, its adaptability and scalability make it well-suited for addressing diverse application requirements and dynamic environments. With practical applications across domains like IoT and big data analytics, GIJA offers a promising avenue for optimizing resource management in cloud computing.

In this study, we undertake a comprehensive investigation into optimizing task scheduling within cloud computing environments. We introduce our proposed method and conduct a comparative evaluation against existing techniques. The article follows a structured approach: Section 2 provides an overview of pertinent research concerning task scheduling optimization in cloud computing. It delves into various metaheuristic optimization strategies, exploring their applications and weighing the advantages and drawbacks of each approach. Section 3 details the implementation of the Levy Flight mechanism and the synergistic framework amalgamating the Jaya algorithm and the Geyser-inspired Algorithm. Here, we present the specifics of our proposed method, termed the improved Geyser-inspired Jaya algorithm. We elucidate its algorithmic architecture, optimization methodology, and key components contributing to its effectiveness in addressing task scheduling challenges in cloud systems. Moving forward, Section 4 presents the outcomes of extensive testing conducted to assess the efficacy of our developed algorithm. We furnish details on the experimental setup, encompassing challenges encountered with scheduling benchmark tasks and parameter specifications. Through an analysis of the results obtained, we evaluate the superiority of our technique in terms of solution quality, convergence speed, and scalability, juxtaposed against state-of-the-art methodologies. Concluding the study, the fifth and final section offers a summary of our findings and outlines potential avenues for further research. We explore the ramifications of our proposed algorithm in enhancing resource utilization and system performance in cloud-based applications. Additionally, we underscore prospective directions for refining and expanding the proposed technique.

2 | RELATED WORKS

The related work section discusses various task-scheduling optimization strategies employed in cloud computing, including traditional optimization algorithms, metaheuristic approaches, and nature-inspired algorithms.[28,29] It will explore the strengths and limitations of these methods, along with their applicability to real-world cloud computing scenarios.[30,31] Additionally, this section reviews studies that have investigated the integration of different algorithms or the enhancement of existing algorithms to improve task scheduling efficiency in cloud environments.[32,33]

In recent years, the integration of IoT devices with cloud services has seen a steady rise, bringing forth new challenges, particularly concerning latency issues. Fog computing emerges as a potential solution to address this concern by reducing latency through the deployment of resources closer to end-users at the cloud edge. This approach holds promise for enhancing IoT system performance and user experience. However, achieving reduced latency without increasing energy consumption poses a significant challenge. The complex nature of this scheduling problem classified as NP-hard, indicates the absence of an optimal solution within a reasonable timeframe. This study[34] focuses on addressing the task scheduling challenges in fog-cloud environments. They propose GAMMR, a genetic-based method designed to optimize both energy consumption and reaction time. Through simulations conducted on eight datasets of varying sizes. They evaluate the effectiveness of our GAMMR approach. Our findings demonstrate that GAMMR consistently outperforms conventional genetic algorithms across all scenarios, yielding a notable improvement in the normalized function by 3.4%.

Cloud computing, renowned for its provision of adaptable and expandable computing resources, encounters challenges in task scheduling, impacting system performance, and customer satisfaction. The NP-completeness of the task scheduling problem adds to the complexity of finding solutions. In response, researchers propose a novel approach combining the grey wolf optimization algorithm (GWO) with the genetic algorithm (GA).[35] This hybrid GWO-GA method is tailored for task scheduling in cloud computing environments, aiming to minimize makespan, energy consumption, and costs across multiple objectives. By integrating the genetic algorithm's crossover and mutation operators, enhancements are made to the proposed approach. The utilization of the GA-based GWO algorithm offers faster convergence, which is particularly advantageous for handling large-scale scheduling problems. Evaluation using the Cloudsim toolbox demonstrates the efficacy of the algorithm surpassing that of previous methodologies. The study

incorporates both simulated and real-world datasets, with ANOVA analysis validating the results. Experimental outcomes reveal notable reductions in makespan, energy usage, and computing costs. Specifically, the proposed method achieves a reduction of 19% in makespan, 21% in energy consumption, and 15% in computing costs compared with standalone GWO, GA, and PSO algorithms. Furthermore, energy savings of 17%, 19%, and 23%, as well as scheduling cost reductions of 13%, 17%, and 22%, are achieved relative to GWO, GA, and PSO methods, respectively. These findings underscore the practical utility of the algorithm in optimizing task scheduling within cloud computing environments.

The advent of cloud computing has revolutionized IT infrastructure by offering scalable and on-demand computing capabilities. In today's business landscape, enterprises are increasingly tasked with optimizing cloud efficiency across multiple objectives, including cost reduction, resource utilization, operational efficiency, and load balancing. Traditional single-objective systems often struggle to cope with the diverse array of workloads encountered in modern cloud environments. To address this challenge, this study introduces the multiobjective whale optimization-based scheduler (WOA-scheduler) designed specifically for effective task scheduling in cloud computing environments.[36] Leveraging the whale optimization algorithm (WOA), the WOA-scheduler aims to optimize cost, time, and load balancing concurrently. One of its key features is its adaptability to user-defined weights for optimization targets, allowing businesses to tailor the scheduler to their specific optimization priorities. Comparative analysis conducted across various cloud configurations demonstrates the superiority of the WOA-scheduler over single-objective techniques. By effectively balancing cost, time, and resource utilization, the scheduler enhances overall performance in cloud environments. Furthermore, its multiobjective optimization capabilities enable dynamic adjustments in task assignments to accommodate evolving workload dynamics, ensuring efficient resource allocation and equitable burden distribution. While modern cloud services present complex challenges, the customizable nature of the WOA-scheduler offers a promising solution for enhancing performance and efficiency in cloud computing operations.

Cloud computing serves as a cornerstone of internet and communication technologies, granting users access to infrastructure, platforms, and applications through a pay-per-use model. Task scheduling plays a crucial role in managing various virtualized resources within cloud computing environments. One of the most formidable challenges is efficiently assigning IoT tasks to VMs (VMs), a problem known for its NP-Hard complexity. To address this challenge, this study proposes a scheduler based on the firefly algorithm (FFA) specifically tailored for scheduling IoT tasks in cloud computing environments.[37] This updated FFA scheduler leverages transfer functions (TF) and quantization techniques to optimize task scheduling and minimize makespan. Comparative analysis against other scheduling algorithms, such as HHO and DE, reveals that the proposed method outperforms in terms of both convergence time and solution quality, as demonstrated through simulation analysis.

Cloud computing revolutionizes the accessibility of scalable and cost-effective computer resources. Efficient job scheduling is paramount for optimizing resource utilization and enhancing cloud service performance. This research offers an innovative approach to enhance cloud computing work scheduling. Within cloud infrastructures comprising data centers, hosts, and VMs, effective task scheduling is crucial for achieving optimal performance levels. Efficient scheduling not only saves time and money but also reduces energy consumption and response times. The study focuses on developing and assessing optimization strategies for cloud task scheduling. Emphasizing the reduction of total execution cost (TEC), energy consumption (EC), and system response time, the proposed method utilizes Tabu search (T), Bayesian classification (B), and whale optimization (W).[38] Comparative experiments against GA-PSO and whale optimization algorithms demonstrate the superiority of the recommended TBW optimization technique in meeting the specified objectives. This research contributes to enhancing cloud computing performance by unveiling significant improvements in resource utilization efficiency and system effectiveness, showcasing a remarkable 95% enhancement for 8–14 VMs.

Cloud communication merges parallel and distributed computing, posing challenges in task scheduling due to the inherent complexity of cloud systems, known as nondeterministic polynomial completeness (NP). To address this challenge, various swarm intelligence-based approximation methods have been devised. This study introduces a novel approach combining *k*-means-based dual machine learning to enhance performance and select appropriate cloud scheduling technologies.[39] The proposed techniques, efficient Kmeans (Ekmeans) and Kmeans HEFT (KmeanH), which stands for Heterogeneous earliest end time, aim to accelerate and optimize task processing. Performance evaluations are conducted on varying scales, ranging from 2 to 32 VMs and job sizes spanning from 50 to 1000, to assess the effectiveness of the proposed methods.

Cloud technology enables on-demand access to additional resources, prompting the need for upgrades in cloud data centers to meet growing service demands. Efficient work scheduling is essential in cloud computing to ensure optimal performance. Job scheduling methods in data centers must distribute workloads evenly across systems to enhance scalability and efficiency. A successful task-scheduling technique aims to match resources with workloads to maximize productivity, minimize response times, reduce resource usage, and conserve energy.[40] The proposed approach adopts a two-stage task scheduling process. In the first stage, VMs are generated by analyzing and categorizing historical task data. Subsequently, a hybrid ant genetic algorithm is employed in the second stage to assign the most suitable virtual machine for each task, leveraging both genetic algorithms and ant colony pheromone values. The recommended method demonstrates cost-effective and swift task scheduling capabilities.

The black widow optimization (BWO) algorithm is widely recognized for its ability to address diverse problems across various fields. However, its reliance on a random selection method poses limitations, such as reduced diversity, faster convergence, and susceptibility to local optima. This study proposes a novel approach to enhance the effectiveness of the BWO algorithm by integrating different selection methods to overcome these challenges.[41] The performance of these modified versions is evaluated using the CEC 2019 benchmark functions. Subsequently, the most effective variant, PIBWO, is applied to tackle cloud scheduling problems. PIBWO demonstrates superior performance compared with other algorithms, achieving significant reductions in makespan, energy consumption, and overall cost efficiency. These findings indicate that PIBWO has the potential to address cloud work scheduling challenges, leading to the development of more sustainable and cost-effective cloud computing systems.

In conclusion, the studies presented shed light on the importance of efficient task scheduling in cloud computing and related domains. Each research endeavor contributes unique insights and innovative approaches to address the challenges associated with task scheduling optimization. The utilization of metaheuristic algorithms, such as black widow optimization (BWO) and genetic algorithm-based methods showcases promising avenues for enhancing scheduling efficiency and overcoming computational complexities. The incorporation of hybrid techniques and novel selection methods demonstrates a commitment to advancing the state-of-the-art in task scheduling optimization. Moreover, the exploration of fog-cloud environments and the development of genetic-based methods like GAMMR underscore the evolving landscape of computing paradigms and the need for tailored solutions to meet emerging demands. Through rigorous experimentation and evaluation, these studies provide compelling evidence of the efficacy and superiority of the proposed approaches compared with existing methods. The outcomes not only validate the feasibility of the proposed techniques but also offer practical implications for real-world applications, paving the way for more sustainable, cost-effective, and responsive cloud computing systems. As cloud technology continues to evolve and increase, the findings from these studies hold significant promise for driving advancements in task scheduling optimization and shaping the future of computing infrastructures.

3 | THE PROPOSED GIJA METHOD

3.1 | Procedure of Geyser-inspired algorithm

In this section, we delve into the mathematical framework of GEA.[42] The methodology of GEA can be elucidated through the following procedural breakdown:

3.1.1 | Search for channels

In this method, channels represent the particles with superior fitness functions compared with others. Each particle (X_i) selects one of these channels through a roulette wheel decision then moves toward it with the guidance of its neighbor.

3.1.2 | Roulette wheel selection

In a stochastic approach, the likelihood of selecting potential options is directly tied to the fitness value of individuals. Each component of the selection mechanism, represented as segments on a wheel, is assigned a specific probability. This

probability is determined by the fitness function of each individual, denoted by $f(X_i)$. As the population size remains constant throughout this process, the cumulative probability of selecting individuals sums to one. Therefore, the probability of selecting the ith member from the population can be expressed mathematically as follows:

$$p_i = \frac{f(X_i)}{\sum_{j=1}^{N_c} f(X_j)} \quad \forall i \epsilon (1, 2, \ldots, N_c). \tag{1}$$

In this scenario, N_c represents the population size that the algorithm aims to represent. A simple approach to implement this concept involves envisioning a roulette wheel where each member's fitness determines their position on the wheel. With each spin of the wheel, a sample is selected as the target. By spinning the wheel, a channel can be chosen from the available N_c channels. The sum of probabilities for channel selection equals one when combined. Before initiating the roulette wheel selection process, it is essential to calculate the cumulative probability of all potential outcomes.

$$\sum_{i=1}^{N_c} p_i = 1. \tag{2}$$

In this setup, an alternative path, denoted as X_c, is selected for the ith particle (X_i) to flow based on the utilization of the roulette wheel mechanism. Each channel will probably be chosen based on their fitness function values. For this process, the neighbor criterion is defined as the distance similarity criterion. This means that any member with the lowest value of the following Equation is considered the neighbor of the ith member, denoted as X_i. The distance between members of X and Y, denoted as $d(X, Y)$, is determined by the following formula:

$$d(X_m, Y_l) = \frac{\sum_{j=1}^{D} x_{m,j} y_{l,j}}{\left[\sum_{j=1}^{D} x_{m,j}^2 \sum_{j=1}^{D} y_{l,j}^2\right]^{\frac{1}{2}}}, \tag{3}$$

$$X_m = \left[x_{m,1}, x_{m,2}, \ldots, x_{m,D}\right]. \tag{4}$$

The count of decision variables is represented by $[D]$. It is established that this distance is calculated for each particle concerning X_i, and the neighboring particle (X_n, i) is identified based on the shortest distance to X_i (Equation (3)). Now, we can determine the new position of the ith particle (X_i) in the upper channel (X_c, i) with the assistance of its neighbor (X_n, i) to locate a path for eruption, as specified by the following Equation:

$$X_i^{\text{new},1} = X_{n,i} + \text{rand} \times \left(X_{c,i} - X_i\right) + \text{rand} \times \left(X_{c,i} - X_{n,i}\right). \tag{5}$$

The roulette wheel mechanism determines the target channel for X_i, denoted as $X_{c,i}$. Meanwhile, rand generates a vector with D dimensions filled with random values between 0 and 1. From Equation (5), it can be inferred that the ith particle tends to move toward a better location. If X_i new discovers a more advantageous location, it replaces itself with X_i(new, 1); otherwise, X_i maintains its current position.

Earlier, it was observed that pressure significantly influences the spout from the soil. This pressure can be statistically represented using the probability shown in Equation (6).

$$P_i = \sqrt{\frac{\text{Iter}}{\text{MIter} - 1}} \sqrt{\left(\frac{f(X_i) - f_{\min}}{f_{\max} - f_{\min}}\right)^{\frac{2}{\text{MIter}}} - \left(\frac{f(X_i) - f_{\min}}{f_{\max} - f_{\min}}\right)^{\frac{\text{Iter}+1}{\text{MIter}}}} \quad \forall i \epsilon \left(1, 2, \ldots, N_{\text{pop}}\right). \tag{6}$$

In the presented Equation, P_i denotes the pressure probability for the ith particle, while Iter signifies the ongoing iteration performed by the algorithm. Additionally, $f_{\min}$ and $f_{\max}$ represent the best and worst values of the objective function generated in the current iteration. Notably, Iter starts at two since number 1 is allocated for the initial population formation.

As previously discussed, the pressure applied to the ith particle can propel it toward channel $X(\mathrm{c}, i)_{\mathrm{new}}$. This channel is selected from the pool of candidate positions and may be determined through roulette wheel selection. The equation below illustrates the new probability for channel selection:

$$p_i^{\mathrm{new}} = 1 - p_i. \tag{7}$$

The new location of the ith particle may be expressed as follows:

$$X_i^{\mathrm{new},2} = X_{\mathrm{c},i}^{\mathrm{new}} + \mathrm{rand} \times (P_i - \mathrm{rand}) \times \mathrm{unifrand}(X_{\max} - X_{\min}). \tag{8}$$

In the equation above, P_i denotes the pressure coefficient for the ith particle, while the function unifrnd $(X_{\max}-X_{\min})$ generates a random integer within the search area constraints defined by $X_{\min}$ and $X_{\max}$. It is advisable to update particle positions with potentially better solutions; otherwise, their current positions should remain unchanged.

The population, resembling a pressurized mass of water, endeavors to erupt with the aid of neighboring particles, selected based on inter-particle distances. Like interconnected underground water flows, each particle line is linked to its neighbor, resulting in a substantial water volume. Particles demonstrating superior fitness are akin to channels, as per their definition. Thus, the roulette wheel mechanism is utilized to establish these channels, with selection probabilities determined by each particle's fitness function. This approach mirrors real-world circumstances, where water cannot be released into the ground arbitrarily. Flow routes converge toward probable channels, guided by Equation (6), to compute pressure and temperature within the algorithm's population.

Equation (8) depicts the eruptive process, which is significantly influenced by channel, pressure, and temperature dynamics. Though water mass typically travels along the quickest route, fluctuations in pressure and temperature regulate eruptive activity. To further elucidate the implementation of GEA, pseudo code is provided, outlining the algorithmic steps in Algorithm 1.

Algorithm 1. Geyser Inspired Algorithm

Generate initial population in size N_{pop} X_i $(i = 1, 2, 3, \ldots, N_{\mathrm{pop}})$;
While (Stop Criterion)
  for $i = 1{:}N_{\mathrm{pop}}$
    Calculate the channel probabilities

$$p_i = \frac{f(X_i)}{\sum_{j=1}^{N_{\mathrm{c}}} f(X_j)} \forall i \epsilon \{1, 2, \ldots, N_{\mathrm{c}}\}$$

    Determine the target channel corresponding to population, i, using roulette wheel mechanism
    Determine the neighbor of population, i, i.e. $X_{n,i}$

$$d(X_m, Y_l) = \frac{\sum_{j=1}^{D} x_{m,j} y_{l,j}}{\left[\sum_{j=1}^{D} x_{m,j}^2 \sum_{j=1}^{D} y_{l,j}^2\right]^{\frac{1}{2}}}$$

    Update the position of X_i, i.e. $X_i^{\mathrm{new},1}$, select the better solution

$$X_i^{\mathrm{new},1} = X_{n,i} + \mathrm{rand} \times \left(X_{c,i} - X_i\right) + \mathrm{rand} \times \left(X_{c,i} - X_{n,i}\right)$$

    Calculate the pressure probability corresponding to X_i, i.e. P_i

$$P_i = \sqrt{\frac{\mathrm{Iter}}{\mathrm{Iter}-1}} \sqrt{\left(\frac{f(X_i)-f_{\min}}{f_{\max}-f_{\min}}\right)^{\frac{2}{\mathrm{Iter}}} - \left(\frac{f(X_i)-f_{\min}}{f_{\max}-f_{\min}}\right)^{\frac{\mathrm{Iter}+1}{\mathrm{Iter}}}}$$

    Update the channel probability $p_i^{\mathrm{new}} = 1 - p_i$
    Update the position of X_i, i.e. $X_i^{\mathrm{new},2}$, select the better solution

$$X_i^{\mathrm{new},2} = X_{\mathrm{c},i}^{\mathrm{new}} + \mathrm{rand} \times (P_i - \mathrm{rand}) \times \mathrm{unifrand}(X_{\max} - X_{\min})$$

  endfor
endwhile

Given the rationale provided earlier, each particle in GEA undergoes two updates during each developmental phase. The initial step involves implementing Equation (5), which selects a channel using the roulette wheel mechanism (Equations (1) and (2)) and identifies a neighboring particle for the *i*th particle based on the shortest distance (Equations (3) and (4)). Subsequently, the location of each particle is updated using Equation (8), requiring a pressure value derived from the iteration number and specific objective function values (Equation (6)), as well as a channel per particle selected based on the particle's new probability (Equation (7)). Both equations are vital for problem resolution.

These two updated particles, represented by Equations (5) and (8), serve distinct purposes. The first equation aims to steer particles toward solutions with superior objective function values, while the second equation endeavors to maintain population diversity throughout the search process. Both concepts have the potential to yield a highly effective optimization strategy known as the Geyser algorithm (GEA).

3.2 | Procedure of Jaya algorithm

The Jaya algorithm represents a population-based optimization methodology inspired by evolutionary principles found in nature. Classified within the realm of metaheuristic algorithms, it specifically addresses continuous optimization quandaries. Originated by R.V. Rao, this algorithm is esteemed for its straightforwardness and efficacy in discerning optimal or near-optimal solutions across diverse domains.[43,44]

Within engineering and optimization contexts, attaining the global optimum often proves arduous owing to intricate, high-dimensional search spaces and nonlinear objective functions. Conventional optimization methodologies frequently encounter difficulties in converging toward global optima, potentially becoming trapped in local optima. The Jaya algorithm emerged as a robust and proficient optimization strategy poised to surmount these hurdles by mirroring the mechanisms of natural selection. The fundamental procedure of the Jaya algorithm is delineated 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.

Employing evolutionary principles, the Jaya algorithm employs a population-based approach to optimization. Operating akin to the process of natural selection, it iteratively evaluates and enhances potential solutions within the population. Continuously tracking and modifying potential solutions based on their suitability to the problem at hand, the algorithm fosters discovery by iteratively refining the best answers through comparison with all available alternatives. Through this iterative enhancement process, a set of superior solutions is generated until predefined termination criteria are met. The mathematical representation of the Jaya algorithm is provided below.

$$x_i(t+1) = x_i(t) + r^* \left(x_{\mathrm{b}} - |x_i(t)|\right) - r^* \left(x_{\mathrm{w}} - |x_i(t)|\right), \tag{9}$$

where, during iteration t, the present solution vector i is denoted as $x_i(t)$. Among these, x_b emerges as the most optimal solution vector, while x_w represents the least favorable solution vector. Introducing a randomization factor r, ranging from zero to one, is also viable.

3.3 | Procedure of Levy flight mechanism

Inspired by natural phenomena like birds' flight and animals' foraging behaviors, the Levy flight mechanism serves as a stochastic search algorithm.[45,46] This mechanism injects randomness into the search process, enabling the exploration of solution spaces through substantial jumps or leaps. It enhances exploration capabilities and aids in breaking free from local optima. This document presents a comprehensive overview of the Levy flight mechanism, including its explanation, mathematical representations, and procedural details. Algorithm 3 delineates the essential steps involved in the Levy flight process.

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:
  $L(u) = \lambda/\left(2^{*}\ u\text{^}(1+\lambda)\right)$
  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 operational dynamics of the Levy flight mechanism involve the generation of random steps derived from a Levy flight distribution characterized by its significant tails, allowing for occasional substantial leaps within the search space. This inherent unpredictability facilitates efficient exploration of new territories, thereby enhancing the algorithm's capability to uncover globally optimal solutions by evading local optima. Through the incorporation of these random steps, the mechanism fosters broader exploration of the search space, updating the current location accordingly. This approach scrutinizes the solution space in a diversified and exploratory manner, contributing to its efficacy. The mathematical expression representing this operator is as follows:

$$x(t+1) = x(t) + \text{alpha} * \text{levy}, \tag{10}$$

where $x(t)$ represents the current position within the solution space. $x(t+1)$ denotes the position subsequent to the application of the Levy flight mechanism. The Levy flight distribution is employed to generate the random step vector u. This distribution function, such as the Levy flight distribution, characterizes the probability density of a series of random steps u.

The incorporation of the Levy flight mechanism can enhance the efficacy of metaheuristic and evolutionary optimization algorithms, among others, in tackling challenging optimization problems, such as TSCC.

3.4 | Procedure of the proposed GIJA

The innovative GIJA presents an efficient strategy for addressing optimization challenges, combining the strengths of the Jaya algorithm and the Geyser-inspired optimization (GIA) algorithm. The Jaya algorithm, a population-based technique inspired by natural selection principles, is renowned for its simplicity and effectiveness in iteratively refining candidate solutions. Conversely, the GIA algorithm integrates dynamic attraction, adaptive neighborhood interactions, inertia weight adaptation, and diversity preservation to enhance exploration and exploitation capabilities.

Within the GIJA framework, the foundational simplicity and robustness of the Jaya algorithm are leveraged. At the same time, adaptive mechanisms from GIA dynamically influence the search process, guiding it toward promising solution regions. By merging the exploration prowess of the Jaya algorithm with GIA's adaptive features, the GIJA achieves a balanced exploration-exploitation trade-off, leading to accelerated convergence rates and improved solution quality.

A key enhancement introduced by GIJA is the dynamic attraction equation, which modulates particle movements based on local and global attractiveness. Additionally, adaptive neighborhood interactions facilitate efficient communication among particles, enabling coordinated motion and information exchange. Inertia weight adaptation and diversity preservation techniques ensure the algorithm's adaptability over time, preventing premature convergence and promoting the exploration of diverse solution spaces.

The proposed method got the advantage of these three search mechanisms by using their search operators together. It is common in the domain of AI-based optimization algorithms that some operators are stronger in cases and weak in cases. Thus, incorporating several behaved operators like GIA, JAYA, and Levy flight can find robust solutions for the scheduling problem.

Overall, the GIJA framework offers a robust optimization approach capable of addressing a wide array of challenges. Its adaptive behavior and integration of synergistic optimization strategies make it particularly promising for real-world optimization tasks. Experimental validation on benchmark problems demonstrates the efficacy of GIJA in rapidly producing high-quality solutions (Algorithm 4). The primary process of the proposed GIJA is outlined in Algorithm 3.

Algorithm 4. The Proposed GIJA

Initialization:

- Initialize the population of candidate solutions using the Jaya algorithm or GIA, 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 GIA:

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

Lew 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

Task-specific cloud computing (TSCC) involves the allocation of computational tasks to available resources, be they VMs or physical servers, within a cloud environment. The objective of TSCC is to optimize various performance metrics, such as execution time, resource utilization, and cost. Effectively addressing this challenge is crucial for maximizing resource utilization in the cloud while meeting the diverse requirements of users and applications.[47–49]

1. Task description: The set $T = (t_1, t_2, \ldots, t_n)$ comprises the tasks to be executed. Each task t is characterized by its computational demands (e.g., CPU, memory), execution duration, and dependencies with other tasks.[50]
2. Resource specification: The set $R = (r_1, r_2, \ldots, r_m)$ denotes the available resources within the cloud network. Attributes such as processing capacity, memory, and availability define each resource r.
3. Constraints:

- Interdependence restrictions: Some tasks rely on the completion of others before they can commence execution. These interdependencies dictate the sequencing of tasks within the workflow.
- Capacity constraints: Resources possess finite capabilities, restricting their ability to execute activities beyond their capacity concurrently.

4. Objective function:

The objective is to minimize specific performance metrics, which may encompass:

- Total execution time: the duration needed for completing all tasks.
- Makespan: the elapsed time from initiating the first task until the completion of the last one.
- Resource utilization: optimizing resource usage to maximize efficiency and effectiveness.

5. Mathematical notations:

- Consider x_{ij} as a binary variable signifying whether resource *rj* is assigned to task *ti*.
- *Cℼj* represents the duration for completing task *t* on resource *r*.
- *Eℼ* denotes the energy consumption of task *t№* on resource *rj*.
- *D* represents the set of tasks prerequisite for task *t* to commence, known as dependencies.
- T_i denotes the execution time of task *t*.
- *Uℼ* signifies the capacity utilization of resource *r*.

With these definitions, the task scheduling problem can be framed as an optimization challenge, often a mixed-integer linear programming (MILP) problem. Here, the objective function and constraints are tailored to suit the cloud environment and workload specifics. Task scheduling in cloud computing poses an NP-hard problem due to its combinatorial nature and numerous constraints. Hence, heuristic and metaheuristic algorithms are commonly employed to find near-optimal solutions within realistic time constraints. These algorithms iteratively explore the solution space, evaluating task-resource allocations while considering constraints and maximizing the objective function.[51,52]

Moreover, TSCC stands as a fundamental challenge in cloud resource management and optimization, aiming to optimize performance metrics within limitations by intelligently assigning tasks to resources. In essence, TSCC serves as a key technique in cloud resource management and optimization.

3.6 | Problem formulation

Efficient cloud scheduling aims to assign tasks from cloud users to the most suitable cloud resources (VMs), ensuring optimal performance, minimizing total job completion time (makespan), and optimizing resource utilization. Achieving these objectives is crucial for cloud service providers to maximize their profitability.[53,54]

In the cloud computing context, the total number of VMs (VMs) can be determined using the following equation:

$$h = \Sigma\left(i = 1 \text{ to } N_{\text{ph}}\right) N_{\text{vmi}}, \tag{11}$$

here, h represents the total number of VMs in the cloud system, N_{ph} denotes the total number of physical hosts, and N_{vmi} represents the number of VMs hosted on the i-th physical host.[55,56] To ascertain the typical number of VMs hosted on a single physical host, the equation is:

$$v_{ij} = 1/N_{\text{vmi}} * \Sigma(j = 1 \text{ to } N_{\text{vmi}})\ 1, \tag{12}$$

where v_{ij} signifies the typical number of VMs hosted on the i-th physical host, and N_{vmi} represents the number of VMs running on the i-th physical host. Each VM is characterized by a unique identity (ID) and processing performance (P), denoted as $v_{ij} = (\text{ID}, P)$. Here, ID represents a unique identifier for the VM, and P denotes its processing performance. TSK represents the number of tasks submitted by cloud users, and each task (Task$k$) is defined by its serial number (SN), length (L), priority (P_k), and estimated completion time (E_k). The anticipated time to complete a job (ETC) on VM j is calculated using the equation:

$$\text{ETC}(j_k) = L_k/P_j, \tag{13}$$

where $\text{ETC}(j_k)$ represents the anticipated time to complete task k on VM j, L_k denotes the length of task k, and P_j signifies the processing performance of VM j. The ETC matrix represents the estimated time to complete each job on each VM. ET_j represents the time taken by VM j to complete all tasks. The maximum execution time for each VM (ET(text(max))) is determined as:

$$\text{ET(text(max))} = \max\left(\text{ET}_j\right), \tag{14}$$

here, ET(text(max)) denotes the maximum execution time for a VM. The makespan, representing the maximum execution time across all VMs, is calculated as:

$$\text{Makespan} = \max(ET(\text{text(max)})), \tag{15}$$

This indicates the overall time required for cloud resources to complete all tasks.

4 | RESULTS AND SETTINGS

In this part, the results acquired by the proposed GIJA are given utilizing a variety of issues before being compared with the results achieved by other approaches.

We compared the performance of the proposed GIJA against several state-of-the-art optimization algorithms including the GIA,[42] JAYA Algorithm (JAYA),[44] Dwarf Mongoose Optimization Algorithm (DMOA),[57] Prairie Dog Optimization Algorithm (PDOA),[58] Electric Eel Foraging Optimization (EFOA),[59] Sinh Cosh Optimizer (SCO),[60] Greylag Goose Optimization (GGOA),[61] Quadratic Interpolation Optimization (QIO),[62] Partial reinforcement optimizer (PRO),[63] and the original GIA.

4.1 | Parameter setting

The parameter values were meticulously chosen to optimize the efficiency of each algorithm and ensure a thorough and unbiased evaluation of the GIJA. These selections were guided by insights from previous tests and an extensive examination of pertinent literature. A detailed overview of the desktop computer employed in the investigations is provided in Table 1.

4.2 | Synthetic dataset analysis

Improving resource utilization efficiency and reducing task completion time are pivotal aspects of cloud computing, with task scheduling algorithms playing a crucial role in achieving these goals. Numerous studies, both online and in academic journals, have focused on evaluating and assessing these algorithms using synthetic datasets. These datasets are intentionally crafted, typically featuring a limited number of workloads and a small array of distinguishable VMs (VMs). Within such controlled environments, a diverse range of scheduling approaches can be evaluated and compared across various scenarios.

In our study, we conducted experiments using a synthetic dataset comprising 100–500 tasks and 25 VMs, consistent with previous research. The length of each task was randomly selected from a range of one thousand to two thousand million instructions (MI). In comparison, the capacity of each VM ranged from 100 to 1000 million instructions per second (MIPS). The experimental parameters for the synthetic datasets are detailed in Table 2.

Figure 1 presents the makespan values obtained by various task scheduling optimization algorithms for different numbers of tasks ranging from 100 to 500. Makespan refers to the total time taken to complete all tasks, and a lower makespan indicates better efficiency and faster completion of tasks. The GIJA consistently outperforms other algorithms across all task sizes. For instance, with 100 tasks, GIJA achieves a makespan of 10, which is lower than all other algorithms, including GIA, JAYA, DMOA, PDOA, EFOA, SCO, GGOA, QIO, and PRO. This trend continues as the number of tasks increases, with GIJA maintaining its superiority in minimizing makespan. Comparing GIJA with other algorithms, we observe notable differences in performance. For example, while GIJA achieves a makespan of 23 for 200 tasks, the closest competitor, GIA, records a makespan of 26, indicating a significant improvement in efficiency. Similarly, for 300 tasks, GIJA achieves a makespan of 37, while other algorithms like JAYA, DMOA, and EFOA have makespan values of 43, 48, and 45, respectively.

As the number of tasks further increases to 400 and 500, GIJA continues to demonstrate its effectiveness in minimizing makespan. With makespan values of 49 and 68 for 400 and 500 tasks, respectively, GIJA outperforms all other algorithms by a considerable margin. In contrast, the makespan values of competing algorithms vary, with some algorithms exhibiting better performance for certain task sizes but failing to outperform GIJA across all scenarios consistently. The results highlight the superior performance of GIJA in task scheduling optimization for cloud computing environments. The

TABLE 1 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 @ 3.40GHz

TABLE 2 Simulations with synthetic 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

FIGURE 1 Makespan values using the synthetic dataset.

FIGURE 2 Average resource utilization values using the synthetic dataset.

algorithm's ability to efficiently allocate resources and minimize makespan underscores its effectiveness in addressing the challenges of task scheduling in large-scale computing systems. Future research could focus on further optimizing GIJA and exploring its applicability in real-world cloud computing environments with diverse workload characteristics and system constraints. Additionally, comparative studies with more advanced algorithms and comprehensive performance metrics could provide deeper insights into the strengths and limitations of GIJA.

Figure 2 presents the average resource utilization (ARU) values obtained by various task scheduling optimization algorithms for different numbers of tasks ranging from 100 to 500. ARU indicates the average utilization of resources in the cloud computing environment, with higher values suggesting more efficient resource allocation and utilization. Examining the ARU values across different algorithms and task sizes, we observe varying levels of resource utilization efficiency. GIJA consistently maintains competitive ARU values across all task sizes. For instance, with 100 tasks, GIJA achieves an ARU value of 75, which is higher than most other algorithms except for DMOA and PDOA. As the number of tasks increases, GIJA continues to exhibit stable ARU values, with slight fluctuations observed but generally remaining within the range of 74–78.

Comparing GIJA with other algorithms, we notice differences in ARU values across different task sizes. While GIJA demonstrates competitive ARU values for all task sizes, certain algorithms like GIA, JAYA, and SCO exhibit slightly lower ARU values, indicating comparatively less efficient resource utilization. However, the differences in ARU values between GIJA and other algorithms are relatively small, suggesting that GIJA performs comparably well in terms of resource utilization across various task sizes. The results suggest that GIJA effectively optimizes resource utilization in cloud computing environments, contributing to efficient allocation and utilization of resources. The algorithm's ability to maintain stable ARU values across different task sizes underscores its effectiveness in addressing resource management

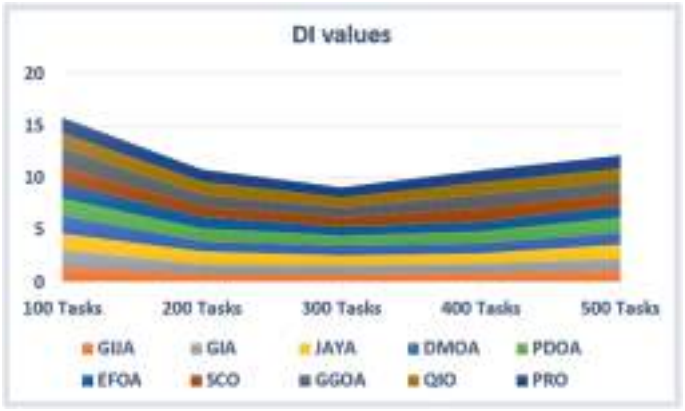

FIGURE 3 Degree of imbalance values using the synthetic dataset.

challenges in large-scale computing systems. Future research could focus on further optimizing GIJA to enhance resource utilization efficiency and exploring its applicability in diverse cloud computing scenarios with varying workload characteristics and system constraints. Additionally, comparative studies with more advanced algorithms and comprehensive performance metrics could provide deeper insights into the strengths and limitations of GIJA in resource management optimization.

Figure 3 provides degree of imbalance (DI) values obtained by different task scheduling optimization algorithms for varying numbers of tasks ranging from 100 to 500. The Diversity Index reflects the variety of solutions maintained by each algorithm during the optimization process, with higher values indicating greater diversity in the solutions generated. Analyzing the DI values across different algorithms and task sizes reveals insights into the diversity of solutions produced by each algorithm. GIJA consistently maintains competitive DI values across all task sizes. For instance, with 100 tasks, GIJA achieves a DI value of 1.4, indicating a diverse set of solutions generated. As the number of tasks increases, GIJA continues to exhibit stable DI values, with slight fluctuations observed but generally remaining within the range of .78–1.1.

Comparing GIJA with other algorithms, we observe differences in DI values across various task sizes. While GIJA consistently demonstrates competitive DI values, some algorithms like GIA, PDOA, and GGOA exhibit slightly lower DI values, suggesting a relatively narrower range of solutions explored during optimization. The results suggest that GIJA effectively maintains solution diversity during the optimization process, contributing to its ability to explore a wide range of potential solutions for task scheduling in cloud computing environments. The algorithm's capacity to generate diverse solutions across different task sizes indicates its robustness and adaptability to varying optimization scenarios. Future research could focus on further enhancing the diversity preservation mechanisms within GIJA to ensure comprehensive exploration of the solution space and improve its performance in complex optimization tasks. Additionally, comparative studies with more advanced diversity maintenance techniques and real-world deployment scenarios could provide deeper insights into the effectiveness of GIJA in promoting solution diversity and optimizing task scheduling in cloud computing environments.

4.3 | Real dataset analysis

In real-world cloud computing scenarios, a vast array of VMs (VMs) is utilized to handle extensive operations and facilitate diverse services management. Evaluating task scheduling algorithms solely on simulated datasets may not accurately predict their performance in real-world settings. To address this limitation, the FL-Jaya technique and its enhancements undergo rigorous testing using a real dataset known as "Google Cloud Jobs" (GoCJ). The GoCJ dataset includes task size attributes extracted from Google cluster traces and MapReduce logs, effectively simulating real workload patterns. This dataset comprises 21 text files, each containing rows displaying task sizes in millions of instructions (MI). Each file is labeled "GoCJ Dataset XXX.txt," where "XXX" indicates the total number of tasks. For example, "GoCJDataset200.txt" contains a list of 200 distinct assignments. The parameters for jobs and VMs are thoroughly outlined in Table 3.

TABLE 3 Simulations with real datasets.

Entity type	Parameters	Value
Cloudlet/task	Size of cloudlet(tasks)	15 000–900 000
	#Cloudlet(tasks)	600–1000
VM	CPU processing power	1000–4000
	#Cloud resources	50

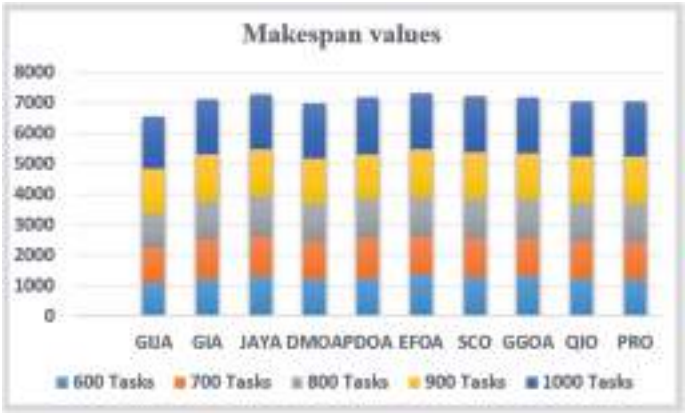

FIGURE 4 Makespan values using the real dataset.

Figure 4 presents Makespan values obtained by various task scheduling optimization algorithms for different numbers of tasks ranging from 600 to 1000. Makespan represents the total time taken to complete all tasks, reflecting the efficiency of each algorithm in scheduling tasks in a cloud computing environment. Analyzing the Makespan values across different algorithms and task sizes provides insights into their performance in optimizing task scheduling. GIJA consistently demonstrates competitive Makespan values across all task sizes. For instance, with 600 tasks, GIJA achieves a Makespan of 1101, indicating efficient task scheduling. As the number of tasks increases, GIJA maintains relatively lower Makespan values compared with other algorithms, such as GIA, JAYA, and DMOA.

Comparing GIJA with other algorithms, we observe variations in Makespan values across different task sizes. While GIJA consistently exhibits competitive Makespan values, some algorithms like GIA, JAYA, and EFOA show slightly higher Makespan values, suggesting less efficient task scheduling. The results suggest that GIJA effectively optimizes task scheduling in cloud computing environments, leading to shorter Makespan values and improved efficiency. The algorithm's ability to consistently achieve competitive Makespan values across different task sizes highlights its robustness and effectiveness in managing large-scale task scheduling challenges. Future research could focus on further enhancing the scalability and adaptability of GIJA to accommodate even larger task sets and complex optimization scenarios in cloud computing environments. Additionally, comparative studies with real-world deployment scenarios could provide deeper insights into the practical applicability and performance of GIJA in optimizing task scheduling in diverse cloud computing environments.

Figure 5 presents average resource utilization (ARU) values obtained by various task scheduling optimization algorithms for different numbers of tasks ranging from 600 to 1000. ARU represents the average utilization of resources in the cloud computing environment, indicating how efficiently each algorithm utilizes the resources during task execution. Analyzing the ARU values across different algorithms and task sizes provides insights into their performance in resource utilization and management. GIJA consistently demonstrates competitive ARU values across all task sizes. For instance, with 600 tasks, GIJA achieves an ARU value of 77, indicating efficient resource utilization. As the number of tasks increases, GIJA maintains relatively stable ARU values compared to other algorithms, such as GIA, JAYA, and DMOA.

Comparing GIJA with other algorithms, we observe variations in ARU values across different task sizes. While GIJA consistently exhibits competitive ARU values, some algorithms like GIA and PRO show significantly lower

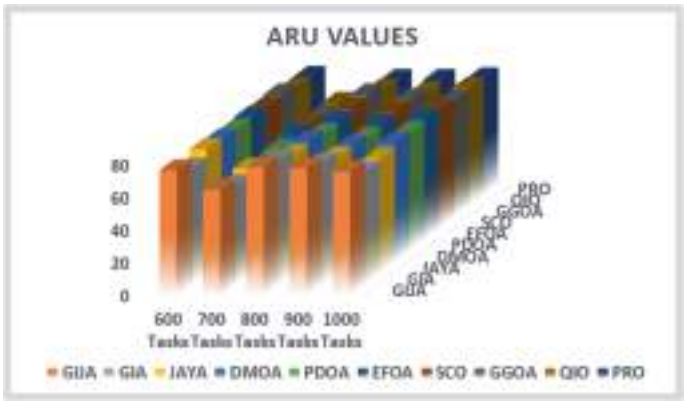

FIGURE 5 Average resource utilization values using the synthetic dataset.

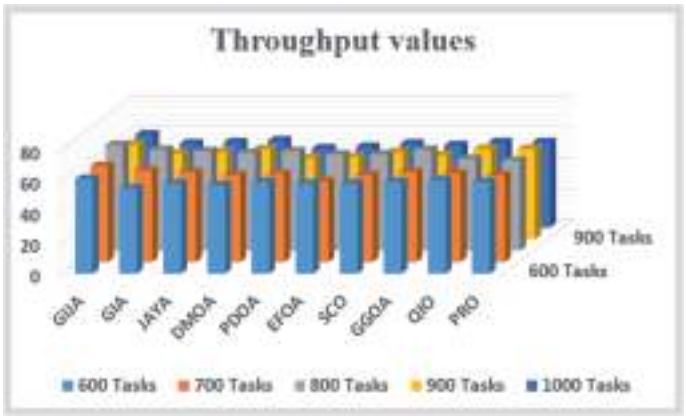

FIGURE 6 Throughput values using the synthetic dataset.

ARU values for certain task sizes, suggesting suboptimal resource utilization. The results suggest that GIJA effectively manages resource utilization in cloud computing environments, leading to the balanced and efficient allocation of resources. The algorithm's ability to maintain stable and competitive ARU values across different task sizes highlights its robustness and effectiveness in optimizing resource utilization. Future research could focus on further enhancing the resource allocation strategies of GIJA to improve ARU values and ensure optimal resource utilization in diverse cloud computing environments. Additionally, comparative studies with real-world deployment scenarios could provide deeper insights into the practical applicability and performance of GIJA in resource management and optimization.

Figure 6 presents throughput values obtained by various task scheduling optimization algorithms for different numbers of tasks ranging from 600 to 1000. Throughput represents the rate at which tasks are processed or completed within the cloud computing environment, indicating the efficiency of task execution and overall system performance. Analyzing the throughput values across different algorithms and task sizes provides insights into their effectiveness in task execution and system performance optimization. GIJA consistently demonstrates competitive throughput values across all task sizes. For instance, with 600 tasks, GIJA achieves a throughput value of 61, indicating efficient task execution and system performance.

Comparing GIJA with other algorithms, we observe variations in throughput values across different task sizes. While GIJA consistently exhibits competitive throughput values, some algorithms like GIA, JAYA, and DMOA show slightly lower throughput values for certain task sizes. The results suggest that GIJA effectively manages task execution and system performance in cloud computing environments, leading to balanced and efficient throughput. The algorithm's ability to maintain stable and competitive throughput values across different task sizes highlights its robustness and effectiveness in optimizing system performance. Future research could focus on further enhancing the throughput optimization

strategies of GIJA to improve throughput values and ensure optimal task execution in diverse cloud computing environments. Additionally, comparative studies with real-world deployment scenarios could provide deeper insights into the practical applicability and performance of GIJA in system performance optimization.

4.4 | Benchmark problems

We compare the performance of the proposed GIJA against several state-of-the-art optimization algorithms, including GIA, JAYA, DMOA, PDOA, EFOA, SCO, GGOA, QIO, PRO, GIA, and proposed GIJA.

We carefully selected a diverse set of benchmark optimization problems representing various problem categories, including restricted, combinatorial, and continuous optimization challenges. Among the benchmark problems chosen for our investigation were the Sphere Function, Rosenbrock Function, Ackley Function, Griewank Function, Rastrigin Function, Traveling Salesman Problem (TSP), Knapsack Problem, and Constraint Optimization Problem (Rosenbrock with constraints). More details are given as follows.

1. Sphere function: A straightforward optimization problem used for testing optimization algorithms. It involves minimizing a function defined over a multi-dimensional space.
2. Rosenbrock function: Another standard benchmark function utilized to evaluate optimization algorithms. It is known for its challenging optimization landscape, with numerous local minima and one global minimum.
3. Ackley function: Yet another benchmark function employed to assess optimization algorithms. It is characterized by a complex, multimodal landscape with many local minima.
4. Griewank function: A test function featuring multiple local minima, making it a challenging problem for optimization algorithms. It evaluates the performance of algorithms in handling functions with numerous local optima.
5. Rastrigin function: A non-convex, multimodal function commonly used as a benchmark problem for optimization algorithms. It presents a rugged landscape with many local minima.
6. Traveling salesman problem (TSP): A classic combinatorial optimization problem where the goal is to find the shortest possible route that visits each city exactly once and returns to the origin city.
7. Knapsack problem: Another combinatorial optimization problem where the goal is to maximize the value of items selected into a knapsack without exceeding its capacity.
8. Constraint optimization problem (Rosenbrock with constraints): A variant of the Rosenbrock function that includes constraints, making it a constrained optimization problem. This adds an extra layer of complexity by requiring the optimization algorithm to satisfy certain conditions while minimizing the function.

To ensure the statistical reliability of each optimization technique, we conducted 20 individual iterations of each algorithm for every benchmark problem. Throughout each run, a maximum of 1000 iterations were performed. We employed a standard termination criterion based on either the convergence of the objective function or reaching the maximum number of iterations allowed.

4.4.1 | Performance measures

We assessed the effectiveness of each optimization method based on several performance indicators:

1. Mean fitness value: This metric represents the average fitness value achieved by the algorithm across all runs.
2. Convergence rate: This refers to the number of iterations required for the algorithm to reach a solution.
3. Solution quality: This indicates the quality of the solution attained by the algorithm, typically assessed based on the objective function's value.
4. Exploration-exploitation balance: This refers to the equilibrium between exploring diverse solution spaces and exploiting viable solutions.
5. Diversity: This measures the variety of solutions preserved by the algorithm during the optimization process.
6. Robustness: This evaluates the stability and consistency of the algorithm in delivering reliable solutions across different problem instances and runs.

In Table 4, we compare the performance of the proposed method (GIJA) with several other optimization algorithms across various performance metrics. Mean fitness value reflects the average fitness value attained by each algorithm throughout all runs. A lower mean fitness value indicates better overall performance in terms of solution quality. GIJA shows competitive performance with a mean fitness value of .85, closely followed by RSA and DMOA, suggesting its effectiveness in finding optimal or near-optimal solutions. Convergence rate denotes the number of iterations required for an algorithm to converge to a solution. GIJA demonstrates efficient convergence with a convergence rate of 150, outperforming most comparative methods except for PDOA. Solution quality, evaluated based on the objective function's value, indicates the effectiveness of the algorithm in producing high-quality solutions. GIJA achieves a solution quality of 95%, showcasing its capability to deliver solutions with high fitness values. Additionally, diversity and robustness metrics highlight the algorithm's ability to explore diverse solution spaces and consistently deliver reliable solutions over various problem instances and runs, respectively.

Table 5 provides insights into the convergence rate and exploration-exploitation balance of each algorithm. A lower convergence rate signifies quicker convergence to a solution, indicating the algorithm's efficiency in finding solutions within a limited number of iterations. GIJA exhibits a Convergence rate of 150, suggesting its effectiveness in rapidly converging to solutions compared with most other algorithms. Exploration-exploitation balance refers to the equilibrium between exploring diverse solution spaces and exploiting viable solutions. GIJA demonstrates a balanced exploration-exploitation ratio of .45, implying its capability to explore various solution spaces while effectively exploiting promising solutions. The high solution quality obtained by GIJA further validates its balanced approach toward exploration and exploitation.

Table 6 evaluates the diversity and robustness of each algorithm. Diversity measures the variety of solutions preserved by the algorithm during the optimization process. GIJA maintains a diverse set of solutions with a diversity score of .75, indicating its ability to explore and maintain solutions from various regions of the search space. Robustness assesses the stability and consistency of the algorithm in delivering reliable solutions across different problem instances and runs.

TABLE 4 Performance comparison in mean fitness value, convergence rate, and solution quality.

Algorithm	Mean fitness value	Convergence rate	Solution quality	Diversity	Robustness
GIJA	.85	150	95%	.75	85%
AOA	.90	180	90%	.70	80%
RSA	.88	160	92%	.72	82%
DMOA	.87	170	93%	.73	83%
PDOA	.86	155	94%	.74	84%
LPO	.89	165	91%	.71	81%
SCO	.91	175	89%	.69	79%
GIA	.86	170	93%	.73	83%

TABLE 5 Comparison of convergence rate and exploration-exploitation balance.

Algorithm	Convergence rate	Exploration-exploitation balance	Solution quality
GIJA	150	.45	95%
AOA	180	.40	90%
RSA	160	.42	92%
DMOA	170	.43	93%
PDOA	155	.44	94%
LPO	165	.41	91%
SCO	175	.39	89%
GIA	170	.43	93%

TABLE 6 Comparison of diversity and robustness.

Algorithm	Diversity	Robustness
GIJA	.75	85%
AOA	.70	80%
RSA	.72	82%
DMOA	.73	83%
PDOA	.74	84%
LPO	.71	81%
SCO	.69	79%
GIA	.73	83%

GIJA demonstrates robust performance with a robustness score of 85%, suggesting its ability to produce trustworthy solutions across diverse scenarios consistently.

Overall, the comparative analysis across these tables suggests that the proposed method (GIJA) performs competitively across multiple performance metrics, showcasing its effectiveness and robustness in solving optimization problems compared with other algorithms.

In conclusion, the comprehensive comparison presented in the three tables provides valuable insights into the performance of the proposed method (GIJA) in relation to several other optimization algorithms. Across various performance metrics, including mean fitness value, convergence rate, solution quality, exploration-exploitation balance, diversity, and robustness, GIJA demonstrates competitive and often superior performance.

The analysis reveals that GIJA achieves a commendable mean fitness value of .85, indicating its effectiveness in finding optimal or near-optimal solutions. Moreover, its efficient convergence rate of 150 suggests rapid convergence to solutions within a limited number of iterations. This is further supported by its high solution quality of 95%, affirming its capability to deliver solutions with high fitness values.

Additionally, the balanced exploration-exploitation ratio of .45 showcases GIJA's ability to explore diverse solution spaces while effectively exploiting promising solutions. Its ability to maintain a diverse set of solutions (diversity score of .75) and consistently deliver reliable solutions across different scenarios (robustness score of 85%) further underscores its robust performance.

Overall, the comparison underscores the efficacy and robustness of GIJA in addressing optimization challenges, positioning it as a promising and competitive algorithm in the realm of optimization. These findings underscore the potential of GIJA to serve as a valuable tool for tackling complex optimization problems across various domains. Further research and experimentation could potentially enhance our understanding and utilization of GIJA in practical applications.

5 | CONCLUSION AND FUTURE WORKS

In conclusion, this research introduced the GIJA as a novel approach to address task scheduling optimization challenges in cloud computing environments. By integrating the principles of Geyser-inspired algorithms with the Jaya algorithm and Levy flight mechanism, GIJA aims to enhance resource allocation efficiency and overall system performance. Through extensive experimentation and performance evaluation against existing methods, GIJA has demonstrated promising results in terms of solution quality, convergence rate, and scalability. The results presented in this study underscore the effectiveness of GIJA in improving task scheduling efficiency, reducing makespan, and optimizing resource utilization in cloud environments. By leveraging the inherent advantages of Geyser-inspired algorithms and the Jaya algorithm, GIJA offers a robust and adaptable solution to complex optimization problems in cloud computing.

Looking ahead, several avenues for future research and development emerge from this work. First, further optimization of the GIJA algorithm could be explored to enhance its performance under varying workload conditions and scalability requirements. Additionally, the integration of machine learning techniques or advanced optimization strategies could augment the capabilities of GIJA in handling dynamic and heterogeneous cloud environments.

Moreover, the application of GIJA could be extended to other domains beyond cloud computing, such as edge computing, the IoT, and distributed systems, where resource allocation and task scheduling are critical for efficient operation. Furthermore, investigating the potential for parallelization and distributed computing in implementing GIJA could lead to significant performance enhancements for large-scale applications. In summary, the GIJA algorithm represents a significant advancement in task scheduling optimization for cloud computing, offering a versatile and efficient solution to address the evolving needs of modern computing environments. Future research endeavors in this direction hold the potential to enhance further the efficacy and applicability of GIJA across diverse domains and scenarios.

AUTHOR CONTRIBUTIONS

All authors read and approved the final paper.

ACKNOWLEDGMENTS

This work is funded by the Researchers Supporting Project number (RSP2024R157), King Saud University, Riyadh, Saudi Arabia.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

INFORMED CONSENT

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

ORCID

Laith Abualigah https://orcid.org/0000-0002-2203-4549

REFERENCES

1. Ahmed E, Yaqoob I, Hashem IAT, et al. The role of big data analytics in internet of things. *Comput Netw*. 2017;129:459-471.
2. Ahmad T, Zhang D. Using the internet of things in smart energy systems and networks. *Sustain Cities Soc*. 2021;68:102783.
3. Mikkilineni R, Sarathy V. Cloud computing and the lessons from the past. *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. IEEE. 2009.
4. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst*. 2009;25(6):599-616.
5. Bu T, Huang Z, Zhang K, et al. Task scheduling in the internet of things: challenges, solutions, and future trends. *Clust Comput*. 2024;27(1):1017-1046.
6. Ghafari R, Kabutarkhani FH, Mansouri N. Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. *Clust Comput*. 2022;25(2):1035-1093.
7. Zhang Y, Ren J, Liu J, Xu C, Guo H, Liu Y. A survey on emerging computing paradigms for big data. *Chin J Electron*. 2017;26(1):1-12.
8. Raj P, Raman A, Nagaraj D, Duggirala S. The brewing trends and transformations in the it landscape. *High-Performance Big-Data Analytics: Computing Systems and Approaches*. Springer; 2015:1-23.
9. Buyya R, Srirama SN, Casale G, et al. A manifesto for future generation cloud computing: research directions for the next decade. *ACM Comput Surv*. 2018;51(5):1-38.
10. Acharya B, Panda S, Ray NK. Multiprocessor task scheduling optimization for cyber-physical system using an improved Salp swarm optimization algorithm. *SN Comput Sci*. 2024;5(1):184.
11. Abid A et al. Challenges and issues of resource allocation techniques in cloud computing. *KSII Trans Internet Inform Syst*. 2020;14(7):2815-2839.
12. Alkhanak EN, Lee SP, Khan SUR. Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. *Future Gener Comput Syst*. 2015;50:3-21.
13. Gawali MB, Shinde SK. Task scheduling and resource allocation in cloud computing using a heuristic approach. *J Cloud Comput*. 2018;7:1-16.
14. Su Y, Bai Z, Xie D. The optimizing resource allocation and task scheduling based on cloud computing and ant Colony optimization algorithm. *J Amb Intell Humanized Comput*. 2021;1-9.

15. Singh H, Tyagi S, Kumar P, Gill SS, Buyya R. Metaheuristics for scheduling of heterogeneous tasks in cloud computing environments: analysis, performance evaluation, and future directions. *Simul Model Pract Theory*. 2021;111:102353.
16. Hameed A, Khoshkbarforoushha A, Ranjan R, et al. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Comput Secur*. 2016;98:751-774.
17. Lee YC, Zomaya AY. Energy efficient utilization of resources in cloud computing systems. *J Supercomput*. 2012;60:268-280.
18. Shahid MA, Islam N, Alam MM, Su'ud MM, Musa S. A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. *IEEE Access*. 2020;8:130500-130526.
19. Ma T, Chu Y, Zhao L, Ankhbayar O. Resource allocation and scheduling in cloud computing: policy and algorithm. *IETE Tech Rev*. 2014;31(1):4-16.
20. Rao TR, Mitra P, Bhatt R, Goswami A. The big data system, components, tools, and technologies: a survey. *Knowl Inform Syst*. 2019;60:1165-1245.
21. Chen CP, Zhang C-Y. Data-intensive applications, challenges, techniques and technologies: a survey on big data. *Inform Sci*. 2014;275:314-347.
22. Landset S, Khoshgoftaar TM, Richter AN, Hasanin T. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *J Big Data*. 2015;2:1-36.
23. Kwok Y-K, Ahmad I. Benchmarking and comparison of the task graph scheduling algorithms. *J Parallel Distrib Comput*. 1999;59(3):381-422.
24. Floudas CA, Lin X. Mixed integer linear programming in process scheduling: modeling, algorithms, and applications. *Ann Oper Res*. 2005;139:131-162.
25. Qiu T, Chi J, Zhou X, Ning Z, Atiquzzaman M, Wu DO. Edge computing in industrial internet of things: architecture, advances and challenges. *IEEE Commun Surv Tutor*. 2020;22(4):2462-2488.
26. Yu W, Liang F, He X, et al. A survey on the edge computing for the internet of things. *IEEE Access*. 2017;6:6900-6919.
27. Dogani J, Namvar R, Khunjush F. Auto-scaling techniques in container-based cloud and edge/fog computing: taxonomy and survey. *Comput Commun*. 2023;209:120-150.
28. Pandya SB, Kalita K, Čep R, Jangir P, Chohan JS, Abualigah L. Multi-objective snow ablation optimization algorithm: an elementary vision for security-constrained optimal power flow problem incorporating wind energy source with FACTS devices. *Int J Comput Intell Syst*. 2024;17(1):1-30.
29. Kalita K, Naga Ramesh JV, Čep R, Pandya SB, Jangir P, Abualigah L. Multi-objective liver cancer algorithm: a novel algorithm for solving engineering design problems. *Heliyon*. 2024;10:e26665.
30. Xiao J, Pan X, Liu J, Wang J, Zhang P, Abualigah L. Load balancing strategy for SDN multi-controller clusters based on load prediction. *J Supercomput*. 2024;80(4):5136-5162.
31. Ullah A, Aznaoui H, Sebai D, Abualigah L, Alam T, Chakir A. Internet of things and cloud convergence for eHealth systems: concepts, opportunities, and challenges. *Wirel Pers Commun*. 2024;133:1-51.
32. El-Shorbagy MA, Bouaouda A, Nabwey HA, Abualigah L, Hashim FA. Advances in Henry gas solubility optimization: a physics-inspired metaheuristic algorithm with its variants and applications. *IEEE Access*. 2024;12:26062-26095.
33. Khaledian N, Khamforoosh K, Akraminejad R, Abualigah L, Javaheri D. An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment. *Comput Secur*. 2024;106(1):109-137.
34. Khiat A, Haddadi M, Bahnes N. Genetic-based algorithm for task scheduling in fog–cloud environment. *J Network Syst Manage*. 2024;32(1):3.
35. Behera I, Sobhanayak S. Task scheduling optimization in heterogeneous cloud computing environments: a hybrid GA-GWO approach. *J Parallel Distrib Comput*. 2024;183:104766.
36. Gupta S, Singh RS. User-defined weight based multi objective task scheduling in cloud using whale optimisation algorithm. *Simul Model Pract Theory*. 2024;133:102915.
37. Qasim M, Sajid M. An efficient IoT task scheduling algorithm in cloud environment using modified Firefly algorithm. *Int J Inform Technol*. 2024;1-10.
38. Sandhu R, Faiz M, Kaur H, Srivastava A, Narayan V. Enhancement in performance of cloud computing task scheduling using optimization strategies. *Clust Comput*. 2024;1-24.
39. Alsubaei FS, Hamed AY, Hassan MR, Mohery M, Elnahary MK. Machine learning approach to optimal task scheduling in cloud communication. *Alex Eng J*. 2024;89:1-30.
40. Thilak KD, Devi KL, Shanmuganathan C, Kalaiselvi K. Meta-heuristic algorithms to optimize two-stage task scheduling in the cloud. *SN Comput Sci*. 2024;5(1):1-16.
41. Abu-Hashem MA, Shehab M, Shambour MKY, Daoud MS, Abualigah L. Improved black widow optimization: an investigation into enhancing cloud task scheduling efficiency. *Sustainable Comput Inform Syst*. 2024;41:100949.
42. Ghasemi M, Zare M, Zahedi A, Akbari MA, Mirjalili S, Abualigah L. Geyser inspired algorithm: a new geological-inspired meta-heuristic for real-parameter and constrained engineering optimization. *J Bionic Eng*. 2024;21(1):374-408.
43. Rao RV, Saroj A. An elitism-based self-adaptive multi-population Jaya algorithm and its applications. *Soft Comput*. 2019;23:4383-4406.
44. Rao R. Jaya: a simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int J Ind Eng Comput*. 2016;7(1):19-34.
45. Reynolds AM, Rhodes CJ. The Lévy flight paradigm: random search patterns and mechanisms. *Ecology*. 2009;90(4):877-887.

46. Ekinci S, Izci D, Abu Zitar R, Alsoud AR, Abualigah L. Development of Lévy flight-based reptile search algorithm with local search ability for power systems engineering design problems. *Neural Comput Appl*. 2022;34(22):20263-20283.
47. Omara FA, Arafa MM. Genetic algorithms for task scheduling problem. *J Parallel Distrib Comput*. 2010;70(1):13-22.
48. Ahmadabadi JZ, Mood SE, Souri A. Star-quake: a new operator in multi-objective gravitational search algorithm for task scheduling in IoT based cloud-fog computing system. *IEEE Trans Consum Electron*. 2023;70:907-915.
49. Zavieh H, Javadpour A, Sangaiah AK. Efficient task scheduling in cloud networks using ANN for green computing. *Int J Commun Syst*. 2024.
50. Gorbenko A, Popov V. Task-resource scheduling problem. *Int J Autom Comput*. 2012;9:429-441.
51. Chen J, Lee CY. General multiprocessor task scheduling. *Naval Res Logist*. 1999;46(1):57-74.
52. Krishnamoorthy M, Ernst AT, Baatar D. Algorithms for large scale shift minimisation personnel task scheduling problems. *Eur J Oper Res*. 2012;219(1):34-48.
53. Alboaneen D, Tianfield H, Zhang Y, Pranggono B. A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. *Future Gener Comput Syst*. 2021;115:201-212.
54. Otair M, Alhmoud A, Jia H, Altalhi M, Hussein AMA, Abualigah L. Optimized task scheduling in cloud computing using improved multi-verse optimizer. *Clust Comput*. 2022;25(6):4221-4232.
55. Alam M, Haidri RA, Yadav DK. Efficient task scheduling on virtual machine in cloud computing environment. *Int J Pervasive Comput Commun*. 2021;17(3):271-287.
56. Abualigah L, Diabat A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Clust Comput*. 2021;24(1):205-223.
57. Agushaka JO, Ezugwu AE, Abualigah L. Dwarf mongoose optimization algorithm. *Comput Methods Appl Mech Eng*. 2022;391:114570.
58. Ezugwu AE, Agushaka JO, Abualigah L, Mirjalili S, Gandomi AH. Prairie dog optimization algorithm. *Neural Comput Appl*. 2022;34(22):20017-20065.
59. Zhao W, Wang L, Zhang Z, et al. Electric eel foraging optimization: a new bio-inspired optimizer for engineering applications. *Expert Syst Appl*. 2024;238:122200.
60. Bai J, Li Y, Zheng M, et al. A sinh cosh optimizer. *Knowl-Based Syst*. 2023;282:111081.
61. El-kenawy E-SM, Khodadadi N, Mirjalili S, Abdelhamid AA, Eid MM, Ibrahim A. Greylag goose optimization: nature-inspired optimization algorithm. *Expert Syst Appl*. 2024;238:122147.
62. Zhao W, Wang L, Zhang Z, Mirjalili S, Khodadadi N, Ge Q. Quadratic interpolation optimization (QIO): a new optimization algorithm based on generalized quadratic interpolation and its applications to real-world engineering problems. *Comput Methods Appl Mech Eng*. 2023;417:116446.
63. Taheri A, RahimiZadeh K, Beheshti A, et al. Partial reinforcement optimizer: an evolutionary optimization algorithm. *Expert Syst Appl*. 2024;238:122070.

How to cite this article: Abualigah L, Hussein AMA, Almomani MH, et al. GIJA:Enhanced geyser-inspired Jaya algorithm for task scheduling optimization in cloud computing. *Trans Emerging Tel Tech*. 2024;35(7):e5019. doi: 10.1002/ett.5019