ICOA: Enhanced Crayfish Optimization for Task Scheduling in Heterogeneous Cloud Environments

Zhiping Zhou
Center of Quality Management, Hebei Software Institute, Baoding City, Hebei Province, 071001, China
E-mail: bdzzp08@126.com

Keywords: cloud computing, task scheduling, crayfish optimization, resource utilization

Received: May 6, 2025

Cloud computing systems are intended to efficiently process large amounts of data and provide on-demand computing resources. Scheduling tasks is one of the intrinsic issues of cloud computing, where tasks are allocated to resources and execution time is reduced while maximizing resource utilization. The present work proposes an improved version of the Crayfish Optimization Algorithm (COA), called ICOA, to solve the task scheduling problem in heterogeneous cloud computing. ICOA integrates chaotic mapping for population diversity, random opposition-based learning (ROBL) for enhanced global exploration, a non-linear control parameter for dynamic search balance, and a Cauchy mutation strategy to avoid premature convergence. Performance of the algorithm is evaluated using CloudSim with NASA, HPC2N, and a synthetic workload. Experimental comparisons on EHHO, MSA–CSA–PAES, ACO, and IWOA demonstrate that the proposed ICOA achieves remarkable improvements: makespan reduction of 50.8%, and resource utilization increase of 26.5%. These experimental results confirm that the proposed ICOA is an efficient and effective solution for task scheduling of complex cloud computing environments.

Povzetek: Za razporejanje nalog v heterogenih oblačnih okoljih je razvita ICOA, izboljšana različica algoritma Crayfish Optimization, ki združuje kaotično inicializacijo, ROBL, nelinearni nadzorni parameter in Cauchyjevo mutacijo.

1 Introduction

Cloud computing is an advanced technology that provides on-demand and elastic access to computer resources on the Internet [1]. It benefits companies and end-users by offering flexibility, cost savings, and operating efficiency through simple data storage, management, and processing [2]. Task scheduling, assigning incoming tasks to available Virtual Machines (VMs), is an essential subproblem in cloud computing. Scheduling tasks properly reduces the makespan (total duration needed to carry out tasks), optimizes the use of available resources, and enhances the overall Quality of Service (QoS) [3]. Inappropriate scheduling causes delays, resource waste, and low system performance and is directly related to operating expenditures and users' satisfaction [4]. Therefore, developing effective scheduling methods is extremely interesting for optimizing cloud computing performance.

Although significant advances have been made, existing scheduling techniques, including metaheuristics and heuristics, are still hindered by inherent challenges in their application to advanced, dynamic clouds. Traditional algorithms are vulnerable to being stuck with local optima too soon, missing better global optima. Numerous algorithms also do not meet an acceptable balance between exploitation (exploring new areas of solutions) and exploration (improving known effective areas), leading to inefficiencies and less-than-optimal scheduling quality. Scalability is also an issue when workloads exceed, further highlighting limitations to adaptability and accuracy in optimizing. These vulnerabilities highlight the need for stronger, scalable, and efficient task scheduling techniques to deliver quality in different clouds.

To overcome these problems, the Improved Crayfish Optimization Algorithm (ICOA) is proposed, which is tailored to scheduling in cloud computing environments. Building on the existing Crayfish Optimization Algorithm (COA), inspired by crayfish's natural foraging activities, ICOA incorporates essential improvements to offer superior global search ability and convergence. Through chaotic mapping for maintaining population diversity, random backward learning for global exploration, non-linear control parameters to promote flexibility between exploration and exploitation, and Cauchy mutation techniques to avoid stagnation, ICOA offers an improved and efficient solution. These improved techniques enhance scheduling performance, makespan, and resource usage within complex clouds. The main contributions of the research are given below:

- Implementation of random reverse learning and chaotic mapping to achieve maximum population diversity and global searching capability.
- Adding a non-linear control parameter to dynamically tradeoff between exploration and exploitation during optimization.
- Implementing a Cauchy mutation strategy to improve local optima escape capability without sacrificing the convergence rate.

- Continued performance assessment of ICOA on real and synthetic datasets to establish its superiority over previous methodologies.

To guide this investigation, we pose the following research questions:

- RQ1: Does chaotic initialization improve convergence speed and task-to-resource mapping efficiency in heterogeneous cloud environments?
- RQ2: How does the combination of non-linear control and Cauchy mutation in ICOA improve the exploration–exploitation balance compared to existing metaheuristic algorithms such as EHHO and ACO?

2 Related work

Jia, et al. [5] proposed an Improved Whale Optimization Algorithm (IWOA), prioritizing significant parameters like task scheduling time, cost, and virtual machine load. They applied an inertial weight strategy to enhance its ability to search locally and to avoid premature convergence. Furthermore, the IWOA has an add-delete operator that advances individual solutions and enhances scheduling quality after each iteration.

Abd Elaziz and Attiya [6] suggested an improved Henry Gas Solubility Optimization algorithm (HGSWC) incorporating the WOA and Comprehensive Opposition-Based Learning (COBL) to enhance cloud task scheduling. While WOA serves to perform local searches to enhance the quality of the solutions, COBL supports weaker solutions by comparing and selecting their opposition. Verified using benchmark functions and compared against traditional HGSO and WOA methods, the proposed HGSWC consistently yielded nearly optimal solutions without excessive computation. Moreover, large-scale simulations using both actual and synthetic workloads confirmed its outperformance compared to six widely used metaheuristics for efficiently tackling the problem of cloud scheduling.

Liu [7] developed an adaptive Ant Colony Algorithm (ACO) to address the shortcomings of conventional ACO, including slow convergence and local optima, in large-scale cloud scheduling. Their strategy initiated an update mechanism for pheromone adaptability, allowing pheromone trails to be dynamically modified to accelerate convergence speed and enhance robustness. Through extensive experimentation, the proposed algorithm consistently yielded well-balanced task execution times, task costs, and system loads in contrast to the standard ACO.

Alsubai, et al. [8] proposed a new task scheduling strategy that integrates the Moth Swarm Algorithm (MSA) and the Chameleon Swarm Algorithm (CSA), together with the Polymorphic Advanced Encryption Standard (PAES), for task scheduling security improvement. The proposed approach considers makespan, resource usage, and bandwidth efficiency while encrypting task information securely. Simulation outcomes across varying task sizes demonstrated improved task scheduling performance, significantly decreasing imbalance, execution time, response delay, and expense. The coupled MSA–CSA–PAES design offered an effective, secure, and energy-efficient approach to task scheduling for complex cloud computing environments.

Malti, et al. [9] introduced a hybrid Flower Pollination Algorithm (FPA)-Grey Wolf Optimizer (GWO) to deal with multi-objective scheduling in heterogeneous clouds. The proposed approach effectively balances exploitation and exploration and prevents convergence and stagnation by utilizing evolutionary crossover operators. CloudSim and sectoral workload simulations reveal improvements in makespan minimization, resource utilization optimization, system balancing, and maximum throughput. A comparative evaluation of the new approach on TSMGWO, GGWO, LPGWO, and standard FPA demonstrated that the hybrid approach effectively handles major scheduling problems.

Pachipala, et al. [10] introduced a two-stage hybrid optimization methodology, known as Blue Updated Jellyfish Search Optimization (BUJSO), that unifies Blue Monkey Optimization (BMO) and Jellyfish Search Optimization (JSO) algorithms. The tasks were scheduled and prioritized using the Analytical Hierarchy Process (AHP) and were partitioned among themselves using k-means clustering. Scheduling considered makespan, utilization cost, migration, and failure risk. Performance Analysis depicted considerable improvements compared to the conventional scheduling methodologies, which indicates the efficacy of the introduced approach in workload balancing, priority, resource allocation, and reducing scheduling costs in various cloud environments.

Fang [11] proposed the Enhanced Harris Hawks Optimization (EHHO) strategy, which is fundamentally aimed at task scheduling optimization for cloud scenarios, utilizing a random walk mechanism. The addition significantly enhanced the algorithm's exploration ability, preventing premature convergence and improving scalability and resource management. Compared to previous techniques, the experimental analysis proved that EHHO efficiently minimized makespan, memory utilization, execution time, and cost.

Despite the significant progress embodied by the reviewed methodologies, several limitations remain, and current cloud task scheduling techniques do not address that. As described in Table 1, most methods have significant issues concerning the effective exploration-exploitation balance, which impacts global search efficiency and solution optimality. In addition to traditional issues such as local optima convergence, stagnation, and low population diversity, the performance of algorithms is significantly hindered, especially for large and highly volatile loads.

Scalability also restricts applicability to realistic scenarios due to increased complexity and the quantity of tasks. The suggested ICOA fills these voids directly through chaotic mapping and random backward learning techniques for enhanced population initialization, non-linear control parameters for preserving exploration and exploitation adaptivity, and Cauchy mutation for preventing stagnation. Thus, it provides better and more efficient scheduling outcomes for adaptive and difficult-to-model cloud instances.

Table 1: Comparative analysis of related work

Reference	Algorithm	Exploration/exploitation balance	Avoiding premature convergence	Main metrics considered	Dataset/testbed used
[5]	Improved whale optimization	Inertial weight strategy	Add-and-delete operator	Time, cost, and VM load	Synthetic workloads
[6]	Henry gas solubility optimization algorithm	WOA local search and COBL	Comprehensive opposition-based learning	Solution quality and makespan	Synthetic and real workloads
[7]	Adaptive ant colony algorithm	Pheromone adaptive update	Dynamic pheromone adjustment	Execution time, load balance, and cost	Synthetic workloads
[8]	Moth swarm algorithm, chameleon swarm algorithm, and polymorphic advanced encryption standard	Swarm integration strategy	Hybrid swarm methods	Makespan, resource utilization, and security	Synthetic workloads
[9]	Hybrid flower pollination–grey wolf	Evolutionary crossover operators	Pollination–wolf search combination	Makespan, throughput, and resource utilization	Synthetic workloads
[10]	Blue updated jellyfish optimization	Analytical hierarchy & clustering	Hybrid optimization methods	Makespan, migration cost, and risk probability	Synthetic workloads
[11]	Enhanced harris hawks optimization	Dynamic random walk strategy	Enhanced exploration capability	Makespan, memory, cost, and scalability	Synthetic workloads
Our work	Improved Crayfish Optimization Algorithm	Non-linear control parameter	Cauchy mutation and ROBL	Makespan and resource utilization	HPC2N, NASA, and synthetic

3 Problem formulation

In a cloud computing environment, task scheduling refers to the optimal assignment of a set of computational tasks to available VMs to enhance system efficiency [12]. The system consists of P Physical Machines (PMs), each running multiple VMs. Given a set of T tasks to be scheduled, the relationship between tasks and VMs is captured by an Expected Execution Time Matrix (EETM), expressed using Eq. 1.

$$EETM = \begin{bmatrix} EETM_{1,1} & EETM_{1,2} & \cdots & EETM_{1,M} \\ EETM_{2,1} & EETM_{2,2} & \cdots & EETM_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ EETM_{T,1} & EETM_{T,2} & \cdots & EETM_{T,M} \end{bmatrix} \quad (1)$$

Elements in EETM denotes the estimated time required for task k to execute on VM m, computed using Eq. 2.

$$EETM_{k,m} = \frac{TaskLength_k}{VM_M IPS_m} \quad (2)$$

Where $TaskLength_k$ is the length (in millions of instructions) of task k and $VM_M IPS_m$ represents the processing speed (million instructions per second) of VM m.

The overall scheduling objectives include minimization of makespan (total completion time) and maximization of Resource Utilization (RU). The makespan stands for the maximum finishing time across all VMs and is calculated using Eq. 3 [13].

$$MKS = \max_{m \in \{1,2,\ldots,M\}} \sum_{k=1}^{T} EETM_{k,m} \quad (3)$$

Resource utilization reflects the efficiency of VM usage over the execution period and is given by Eq. 4 [14].

$$RU = \frac{\sum_{k=1}^{T} ExecTime_{VM_m}}{Makespan \times M} \quad (4)$$

Where $ExecTime_{VM_m}$ is the total time VM m spends executing its assigned tasks.

To evaluate candidate scheduling solutions, a weighted sum fitness function is used that combines the two conflicting objectives: minimizing makespan and maximizing resource utilization. Both objectives are first normalized using min–max scaling over observed ranges during initialization. The combined fitness function is defined as:

$$F = w_1 \cdot \frac{MKS_{max} - MKS}{MKS_{max} - MKS_{min}} + w_2 \cdot \frac{RU - RU_{min}}{RU_{max} - RU_{min}} \quad (5)$$

Where w_1 and w_2 are the weights assigned to makespan and resource utilization respectively (with $w_1 + w_2 = 1$). In our experiments, we used $w_1 = 0.6$ and $w_2 = 0.4$, giving slightly more importance to makespan reduction.

The overall goal is to determine a task-to-VM allocation that minimizes the total completion time while maximizing system resource efficiency, thus enhancing the performance and scalability of cloud computing operations.

4 Proposed methodology

4.1 Basic crayfish optimization algorithm

COA is a nature-inspired metaheuristic algorithm modeled on crayfish's foraging, competition, and summer resort (retreat) behaviors. It alternates between exploration (diversifying search) and exploitation (refining search) phases depending on the environmental conditions, simulated via a random temperature parameter [15]. The

algorithm defines a colony of crayfish C, where each crayfish represents a candidate solution in the optimization space.

In a multi-dimensional search space, each crayfish individual C_i is a $1 \times D$ vector, where D denotes the number of problem dimensions. The initial population is randomly generated as follows:

$$C = [C_1, C_2, \ldots, C_N] \tag{6}$$

$$C_{i,j} = LB_j + \left(UB_j - LB_j\right) \times rand \tag{7}$$

Where LB_j and UB_j are the lower and upper bounds for dimension j, $rand$ is a random number between 0 and 1, and N is the population size.

The environmental temperature $Temp$ regulates the behavioral mode (exploration or exploitation) of crayfish calculated using Eq. 8.

$$Temp = 15 \times rand + 20 \tag{8}$$

The food intake under different temperatures is modeled by a normal distribution using Eq. 9.

$$Intake = C_1 \times \frac{1}{\sqrt{2\pi\sigma}} \times exp\left(-\frac{(Temp - \mu)^2}{2\sigma^2}\right) \tag{9}$$

Where μ is the optimal feeding temperature, σ controls the spread, and C_1 is a scaling coefficient.

When $Temp > 30°C$ and $rand < 0.5$, crayfish seek cooler environments. They move toward a temporary refuge (cave) located at:

$$Cave = \frac{C_{best} + C_{local}}{2} \tag{10}$$

Where C_{best} is the best individual found so far and C_{local} is the current best of the population.

Crayfishes are updated using Eq. 11.

$$C_{i,j}^{t+1} = C_{i,j}^{t} + \lambda \times rand \times \left(Cave - C_{i,j}^{t}\right)$$
$$\lambda = 2 - \left(\frac{t}{T}\right) \tag{11}$$

Where t is the current iteration and T is the maximum number of iterations.

If $Temp > 30°C$ and $rand \geq 0.5$, multiple crayfish compete for the cave. The new position is determined by Eq. 12.

$$C_{i,j}^{t+1} = C_{i,j}^{t} - C_{z,j}^{t} + Cave$$
$$z = round\left(rand \times (N-1)\right) + 1 \tag{12}$$

Where z is a randomly selected individual and $rand$ controls random competition selection.

When $Temp \leq 30°C$, crayfish engage in foraging activities. The food source corresponds to the best solution so far.

$$Food = C_{best} \tag{13}$$

The size of the food determines crayfish behavior as follows:

$$Size = C_3 \times rand \times \left(\frac{fitness_i}{fitness_{Food}}\right) \tag{14}$$

Where C_3 is a food scaling factor, $fitness_i$ is the fitness of crayfish i, and $fitness_{Food}$ is the fitness at the food location.

If $Size > \frac{C_3+1}{2}$ (i.e., the food is too large), the food is shredded using Eq. 15.

$$Food_{new} = exp\left(-\frac{1}{Size}\right) \times Food \tag{15}$$

Then, crayfish update their positions using alternating sine-cosine foraging behavior:

$$C_{i,j}^{t+1} = C_{i,j}^{t} + Food_{new} \times Intake \times (cos(2\pi \times rand) - sin(2\pi \times rand)) \tag{16}$$

In cases $Size \leq \frac{C_3+1}{2}$ (manageable food size), the crayfish directly migrate toward the food:

$$C_{i,j}^{t+1} = \left(C_{i,j}^{t} - Food\right) \times Intake + Intake \times rand \times C_{i,j}^{t} \tag{17}$$

4.2 Improved crayfish optimization algorithm

The basic COA demonstrates considerable potential but faces premature convergence, uneven initial population distribution, and suboptimal exploration-exploitation balance. To address these challenges, several enhancements are introduced in the ICOA.

Random initialization in COA can lead to poor population diversity, reducing global search ability and causing early stagnation. To counteract this, a chaotic sequence-based initialization is employed using logistic mapping, which enhances diversity and ensures more uniform coverage of the search space. The logistic mapping is defined as:

$$X_{n+1} = \mu \times X_n \times (1 - X_n) \tag{18}$$

Where X_n is the current value and μ is the control parameter.

In this study, we selected μ = 2.5 for the logistic mapping function to balance diversity and stability in the initialization phase. While higher values of μ (e.g., >3.5) increase chaos and exploration, they may also destabilize early convergence in constrained scheduling problems. Choosing μ = 2.5 offers a moderate level of chaos, sufficient to diversify the initial population without introducing excessive randomness or instability. This choice is supported by recent applications in UAV path planning that successfully use similar values

ICOA boosts exploration potential and resilience against local optima by generating initial positions through logistic chaos.

Random Opposition-Based Learning (ROBL) is incorporated to further enrich population diversity. Instead of merely randomizing individuals, each candidate X_i generates an opposite solution $\widehat{X_i}$ based on random perturbations:

$$\widehat{X_i} = L_j + U_j - rand \times X_i \tag{19}$$

Where L_j and U_j are the lower and upper bounds of the search space, and $rand \in [0, 1]$ is a uniformly sampled random vector that introduces controlled perturbation. Unlike classical opposition learning, ROBL allows each dimension to be perturbed by a random scaling factor, enabling non-symmetric and domain-aware reflection.

To ensure feasibility within constrained domains (e.g., task–VM index ranges), the generated X_i is projected back into the search space using Eq. 20.

$$\hat{X}_i^{corrected} = min\left(max\left(\hat{X}_i, L_j\right)U_j\right) \quad (20)$$

The better of the original and opposite solution (in terms of fitness) is retained, promoting better exploration and faster convergence during early iterations.

The original COA employs a linear decreasing control factor, which may cause inefficient transitions between global search (exploration) and local search (exploitation). ICOA introduces a non-linear control parameter to better adapt search dynamics during iterations. The update mechanism is modified as follows:

$$\lambda = 2 - exp\left(-\frac{t^2}{T \times rand}\right) \quad (21)$$

Non-linear control parameters dynamically control the balance between exploration and exploitation when optimizing. Compared to linear decay (which linearly decreases as time increases), the exponential form enables rapid initial exploration and gradually escalates exploitation as the search proceeds. This adaptive response is better for high-dimensional, dynamic scheduling problems.

The solution update during exploration (e.g., summer resort stage) becomes:

$$C_{i,j}^{t+1} = C_{i,j}^t + \lambda \times rand \times \left(Cave - C_{i,j}^t\right) \quad (22)$$

This dynamic control accelerates exploitation when needed and prolongs exploration when beneficial, improving convergence precision.

During later optimization stages, populations often risk becoming trapped in local optima. To mitigate this, ICOA employs a Cauchy mutation strategy to perturb the best-found solution. The Cauchy mutation is modeled using Eq. 23.

$$Y_{new} = Y_{best} \times \left(1 + f_{Cauchy}(0,1)\right) \quad (23)$$

Where $f_{Cauchy}(0,1)$ generates a random value following a Cauchy distribution (centered at 0 with a scale of 1).

The Cauchy distribution is selected for its heavy-tailed property, which allows for larger random jumps compared to the Gaussian distribution. This makes it especially effective in escaping local optima during later stages of optimization, when the population tends to cluster. In contrast, Gaussian mutations are more conservative and may not provide sufficient perturbation to overcome local stagnation.

If the mutated solution Y_{new} has better fitness, it replaces Y_{best}; otherwise, the original is retained:

$$Y_{best} = \begin{cases} Y_{best}, & if\ f(Y_{best}) < f(Y_{new}) \\ Y_{new}, & otherwise \end{cases} \quad (24)$$

This mechanism enables ICOA to escape from local minima dynamically and maintain search momentum. The proposed ICOA effectively addresses the primary limitations observed in conventional metaheuristic-based task scheduling methods. Algorithm 1 and Figure 1 present the pseudocode and flowchart of ICOA, respectively. Using logistic chaotic mapping during the initialization phase, ICOA ensures population diversification and distribution within the search space, resisting the possibility of early convergence due to poor initial population configurations.

Further, by applying ROBL, population diversification is maintained by generating opposite candidate solutions and selecting improved individuals. This two-pronged tactic decisively enhances the algorithm's global exploration ability to effectively traverse challenging optimization topologies and prevent early convergence into local optima. Lastly, implementing the non-linear control parameter dynamically controls the exploration and exploitation balance according to the number of iterations, enabling ICOA to exhibit persistent search flexibility during optimization.

By applying the Cauchy mutation principle, ICOA improves population diversification and search flexibility. Through controlled perturbations to the best-so-far solutions, Cauchy mutation reduces the likelihood of premature convergence during the subsequent optimization stages, a characteristic weakness of traditional COA and other swarm-based techniques. An exponential decrease of the control parameter ensures that the intensification of search is adequately fortified along the progress of search without hindering first-stage

Algorithm 1 Pseudocode of ICOA

Input:
Set of tasks (T), set of VMs (M), population size (N), problem dimension (D), and maximum number of iterations (T_{max})
Initialize:
Generate the initial population using Eq. 6
Apply ROBL to enhance diversity (Eq. 19)
Evaluate the fitness of all individuals
While ($t < T_{max}$):
For each individual C_i in C:
  Calculate environmental temperature using Eq. 8
  If $Temp > 30$:
    If $rand < 0.5$:
      //Summer resort stage (Eq. 10)
      Update position using Eq. 11
    Else:
      // Competition stage (Exploitation)
      Select a random individual C_z
      Update position using Eq. 12
  Else:
    // Foraging stage (Exploitation)
    Food = C_{best}
    Calculate food size using Eq. 14
    If $Size > \frac{C_3+1}{2}$:
      Update food position using Eq. 15
      Update position using Eq. 16
    Else:
      Direct movement towards food using Eq. 17
End For
Apply Cauchy mutation:
  Mutate the best individual using Eq. 21
  If fitness(C_{new}) better than fitness(C_{best}), replace C_{best}
Update the best solution found so far
t=t+1
End While
Output:
Best solution

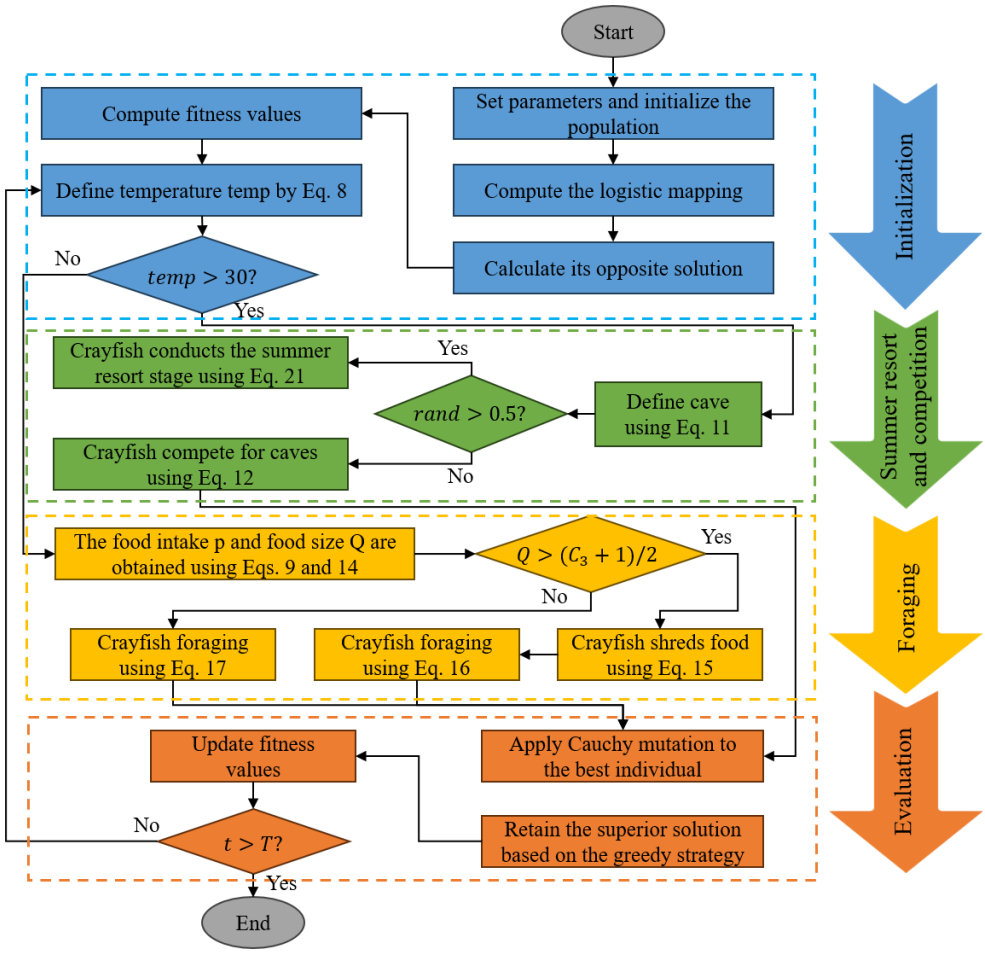

Figure 1: Flowchart of ICOA

exploration capability. These optimizations, together, on an individual basis, optimize and improve task schedule quality under cloud environments by decreasing makespan, optimizing resource utilization, and encouraging resilience for dynamic and large-scale scheduling cases.

5 Results and discussion

This section provides an overall assessment of the ICOA regarding scheduling effectiveness for various cloud computing workloads. The experimental framework was deployed using the CloudSim simulation package, with three benchmark datasets, including HPC2N, NASA, and simulated datasets. These datasets were chosen to simulate various workload complexities so that ICOA's performance could be thoroughly evaluated under real-world and synthetic setups. The problem of scheduling was considered using independent and non-preemptive tasks to model realistic cloud environments. Every experimental setup was executed 30 times to achieve statistical accuracy, and there were two primary performance metrics: makespan and resource utilization.

In our CloudSim-based simulation environment, we deployed 50 virtual machines (VMs) evenly distributed across three heterogeneous types: Type A (1 vCPU @ 1000 MIPS, 2048 MB RAM), Type B (2 vCPUs @ 1500 MIPS, 4096 MB RAM), and Type C (4 vCPUs @ 2000 MIPS, 8192 MB RAM). Task lengths were generated using a uniform distribution over the range [1000, 10000] million instructions (MI), simulating both lightweight and computationally intensive workloads. A space-shared VM scheduling policy with non-preemptive tasks was applied. To ensure the robustness of results, each experiment was repeated 30 times, and all stochastic processes, including those in CloudSim and ICOA, were initialized using a fixed random seed (seed = 42) for reproducibility.

The performance of ICOA was compared with four well-known algorithms: EHHO, MSA–CSA–PAES, ACO, and IWOA. As illustrated in Figures 2-7, simulation results indicate that ICOA surpassed the compared algorithms in all cases. For instance, when the tasks were 2500, ICOA achieved the highest makespan reductions of 13.9% in the case of the HPC2N dataset, 50.8% in the case of the NASA dataset, and 34.7% in the case of the simulated dataset in comparison to other competing algorithms. These improvements are attributed to ICOA's adaptive exploration-exploitation balance, achieved through temperature-driven behavioral phases, chaotic initialization, and dynamic non-linear control parameters.

In resource utilization terms, as reflected in Figures 5–7, ICOA realized gains of as much as 15.7% for the HPC2N dataset, 17.7% for the NASA dataset, and 26.5% for the simulated dataset relative to the following best-performing algorithm under the scenario of 2500 tasks.

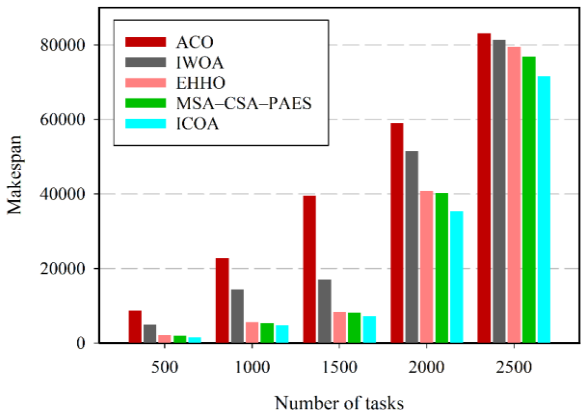

Figure 2: Makespan results for the HPC2N dataset

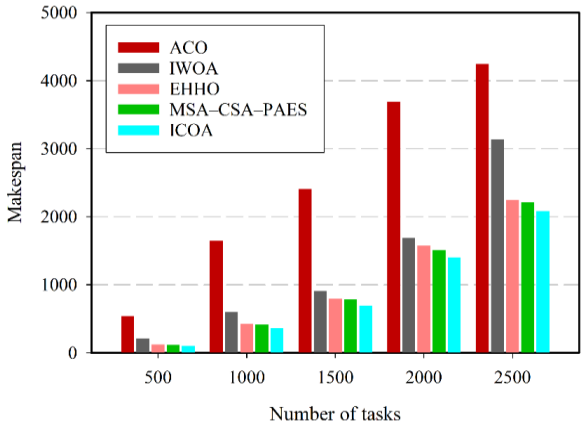

Figure 3: Makespan results for the NASA dataset

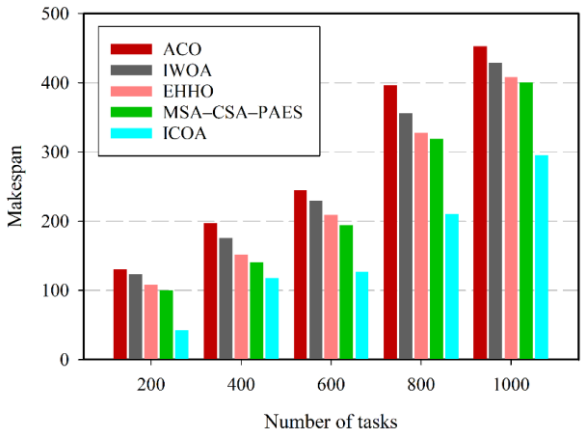

Figure 4: Makespan results for the simulated dataset

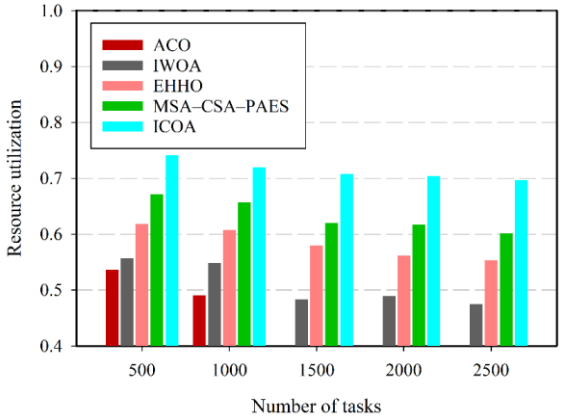

Figure 5: Resource utilization results for the HPC2N dataset

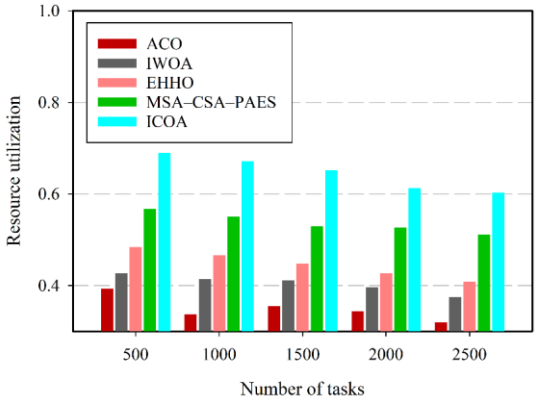

Figure 6: Resource utilization results for the NASA dataset

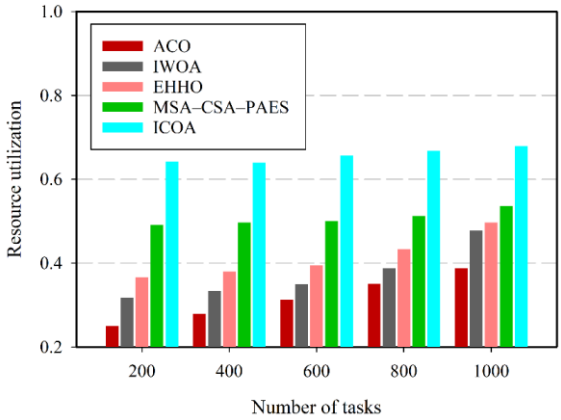

Figure 7: Resource utilization results for the simulated dataset

These improvements reflect the ability of ICOA to find optimal task-to-resource mappings through chaotic sequences when populating the population and improving them using Cauchy mutation and greedy selection policies. Incorporating non-linear decay into the control parameters enabled ICOA to achieve reasonable convergence during the simulation phases, leading to improved load balancing and greater utilization rates. Overall, these results attest that ICOA expedites task scheduling and optimizes resource consumption better than the baseline algorithms.

To evaluate the runtime efficiency of ICOA, we compared the average execution time (in seconds) and approximate floating-point operation counts (FLOPs) against baseline algorithms (EHHO, ACO, IWOA, and MSA–CSA–PAES) on the HPC2N dataset with 2500

tasks and 50 VMs. Results (summarized in Table 2) show that while ICOA has a slightly higher computational cost than ACO and EHHO due to the integration of Cauchy mutation and ROBL, it remains significantly more efficient than MSA–CSA–PAES. Notably, ICOA achieved a 36.5% faster runtime than MSA–CSA–PAES and comparable or better performance in FLOPs per solution.

An ablation study was performed on the HPC2N dataset with 2500 tasks. We defined five variants of ICOA by selectively removing one component at a time from the full algorithm:

- ICOA–C: without chaotic mapping
- ICOA–R: without ROBL
- ICOA–L: using a linear control parameter instead of λ
- ICOA–M: without Cauchy mutation
- Full ICOA: all components enabled

The performance of each variant was evaluated over 30 independent runs. Table 3 show that each component contributes meaningfully, with the non-linear control parameter and Cauchy mutation having the most significant effects on makespan and convergence robustness.

Table 2: Runtime and FLOPs comparison

Algorithm	Avg. runtime (s)	Approx. FLOPs	Normalized score (Runtime × FLOPs)
ACO	11.2	1.3×10^6	1.46×10^7
EHHO	13.4	1.6×10^6	2.14×10^7
IWOA	15.8	2.2×10^6	3.48×10^7
MSA–CSA–PAES	21.6	2.5×10^6	5.40×10^7
ICOA	13.7	1.9×10^6	2.60×10^7

Table 3: Ablation study results

Variant	Avg. makespan (ms)	Resource utilization (%)	Std. Dev (Makespan)
ICOA–C	54.2	74.4	3.6
ICOA–R	53.5	75.1	3.3
ICOA–L	55.6	73.7	3.8
ICOA–M	56.9	72.2	4.1
Full ICOA	51.1	77.8	2.1

6 Discussion

Our experimental results indicate that ICOA significantly outperforms baseline algorithms for makespan minimization and resource utilization. Compared to EHHO and ACO, ICOA has the advantage of higher convergence speed, which is significantly true for large-scale jobs. This is because of the non-linear control parameter, where the algorithm can gradually transition from exploration to exploitation more adaptively, avoiding premature confinement.

In contrast to MSA–CSA–PAES and IWOA, ICOA exhibits higher robustness and scalability. While the multi-swarm technique of MSA–CSA–PAES coupled with encryption methods increases computational overhead due to added complexity, the lightweight enhancements of ICOA, that is, chaotic initialization for initial diversification of the population and late-injection of controlled randomness by Cauchy mutation, offset complexity against performance. Its successful applications for other optimization problems, such as UAV path planning, inspired these capabilities.

Chaotic mapping boosts initial search coverage, reducing the chances of suboptimal clustering. This is a complement to Cauchy mutation, which injects rare long jumps into solution space, enabling the algorithm to escape from locally optimum points without disrupting convergence habits. With these features, ICOA can potentially be applied to resource allocation problems other than cloud computing, such as robot path planning, manufacturing scheduling, and dynamic load balancing. It allows easy modification by replacing the fitness evaluation model for application-defined objectives.

7 Conclusion

In the present study, ICOA is proposed as a solution for task scheduling within the cloud computing framework. A range of improvements to the underlying COA has been proposed to handle common issues of premature convergence, exploration restrictions, and ineffectiveness in large-scale optimization. For diversity maximization, the initiation of a chaotic sequence from logistic mapping has been applied, and the application of ROBL has also enhanced the capacity of global search. Adaptive balancing of exploration and exploitation has been achieved by applying a non-linear control parameter, and the Cauchy mutation has been applied for efficient escape from the optima of subsequently iterated values.

Extensive experimental evaluation was performed using realistic and synthetic HPC2N, NASA, and Synthetic benchmarks. The experiments illustrate that ICOA significantly outperforms the existing benchmarking algorithms, namely EHHO, MSA–CSA–PAES, ACO, and IWOA, for makespan and resource utilization. Utilizing adaptive exploration techniques along with better exploitation techniques, ICOA gained better convergence, better scheduling efficiency, and better scalability. These findings validate that ICOA is a robust and practical approach to task scheduling optimization processes using dynamic and heterogeneous cloud settings.

Beyond theoretical improvements, ICOA demonstrates practical feasibility for real-world scheduling environments. With an average execution time of under 14 seconds for 2500 tasks on a standard desktop machine, the algorithm is suitable for integration into dynamic cloud platforms. ICOA's modular structure and compatibility with Java-based simulation (CloudSim) make it readily portable to open-source cloud management systems such as OpenStack, where its scheduling logic can be embedded within Nova scheduler extensions. It is also applicable to AWS-based testing environments through workload emulation tools like EC2Sim or CloudSched. These characteristics position ICOA not just as an academic tool, but as a deployable metaheuristic suitable for heterogeneous and on-demand cloud infrastructure.

References

[1] C. Ji, J. Zhou, and F. Kong, "Optimization of prediction intensity of big data clustering algorithm integrated with distributed computing in cloud environment," *Informatica,* vol. 48, no. 20, 2024, doi: https://doi.org/10.31449/inf.v48i20.6054.

[2] H. Wang, K. J. Mathews, M. Golec, S. S. Gill, and S. Uhlig, "AmazonAICloud: proactive resource allocation using amazon chronos based time series model for sustainable cloud computing," *Computing,* vol. 107, no. 3, p. 77, 2025, doi: https://doi.org/10.1007/s00607-025-01435-w.

[3] V. Hayyolalam, B. Pourghebleh, M. R. Chehrehzad, and A. A. Pourhaji Kazem, "Single-objective service composition methods in cloud manufacturing systems: Recent techniques, classification, and future trends," *Concurrency and Computation: Practice and Experience,* vol. 34, no. 5, p. e6698, 2022, doi: https://doi.org/10.1002/cpe.6698.

[4] O. K. J. Mohammad, M. E. Seno, and B. N. Dhannoon, "Detailed Cloud Linear Regression Services in Cloud Computing Environment," *Informatica,* vol. 48, no. 12, 2024, doi: https://doi.org/10.31449/inf.v48i12.6771.

[5] L. Jia, K. Li, and X. Shi, "Cloud computing task scheduling model based on improved whale optimization algorithm," *Wireless Communications and Mobile Computing,* vol. 2021, no. 1, p. 4888154, 2021, doi: https://doi.org/10.1155/2021/4888154.

[6] M. Abd Elaziz and I. Attiya, "An improved Henry gas solubility optimization algorithm for task scheduling in cloud computing," *Artificial Intelligence Review,* vol. 54, no. 5, pp. 3599–3637, 2021, doi: https://doi.org/10.1007/s10462-020-09933-3.

[7] H. Liu, "Research on cloud computing adaptive task scheduling based on ant colony algorithm," *Optik,* vol. 258, p. 168677, 2022, doi: https://doi.org/10.1016/j.ijleo.2022.168677.

[8] S. Alsubai, H. Garg, and A. Alqahtani, "A novel hybrid MSA-CSA algorithm for cloud computing task scheduling problems," *Symmetry,* vol. 15, no. 10, p. 1931, 2023, doi: https://doi.org/10.3390/sym15101931.

[9] A. N. Malti, M. Hakem, and B. Benmammar, "A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems," *Cluster Computing,* vol. 27, no. 3, pp. 2525–2548, 2024, doi: https://doi.org/10.1007/s10586-023-04099-3.

[10] Y. Pachipala, D. B. Dasari, V. V. R. M. Rao, P. Bethapudi, and T. Srinivasarao, "Workload prioritization and optimal task scheduling in cloud: introduction to hybrid optimization algorithm," *Wireless Networks,* pp. 1–20, 2024, doi: https://doi.org/10.1007/s11276-024-03793-3.

[11] W. Fang, "Enhanced Task Scheduling Algorithm Using Harris Hawks Optimization Algorithm for Cloud Computing," *International Journal of Advanced Computer Science & Applications,* vol. 16, no. 1, 2025, doi: https://dx.doi.org/10.14569/IJACSA.2025.0160189.

[12] J. Ma, C. Zhu, Y. Fu, H. Zhang, and W. Xiong, "Cloud Computing Resource Scheduling Method Based on Optimization Theory," *Informatica,* vol. 48, no. 23, 2024, doi: https://doi.org/10.31449/inf.v48i23.6901.

[13] N. Alruwais *et al.*, "Farmland fertility algorithm based resource scheduling for makespan optimization in cloud computing environment," *Ain Shams Engineering Journal,* vol. 15, no. 6, p. 102738, 2024, doi: https://doi.org/10.1016/j.asej.2024.102738.

[14] R. Sandhu, M. Faiz, H. Kaur, A. Srivastava, and V. Narayan, "Enhancement in performance of cloud computing task scheduling using optimization strategies," *Cluster Computing,* vol. 27, no. 5, pp. 6265–6288, 2024, doi: https://doi.org/10.1007/s10586-023-04254-w.

[15] H. Jia, H. Rao, C. Wen, and S. Mirjalili, "Crayfish optimization algorithm," *Artificial Intelligence Review,* vol. 56, no. Suppl 2, pp. 1919–1979, 2023, doi: https://doi.org/10.1007/s10462-023-10567-4.