P-ISSN 1991-8941
E-ISSN 2706-6703
Vol. 20, No. 1 (2026)

Journal of University of Anbar for Pure Science
(JUAPS)

journal homepage: juaps.uoanbar.edu.iq

Open Access

Original Paper

An adaptive mountain gazelle optimizer for minimizing process waiting time in multi-core systems

Ali Qasem Ahmed [1*], **Muntadher Khamees** [1], **Khalid Shaker** [2]

[1]Department of Computer Science, College of Science, University of Diyala, Baquba, Iraq
[2]College of Computer Science and Information Technology, University of Anbar, Iraq

ARTICLE INFO

Received: 30/05/2025
Accepted: 12/08/2025
Available online: 24/03/2026
April Issue
10.37652/juaps.2025.160843.1400

Cite @ JUAPS

Corresponding author
Ali Qasem Ahmed
scicompms232409@uodiyala.edu.iq

ABSTRACT

Optimizing CPU task scheduling is crucial for increasing system performance, particularly in multi-core environments. Traditional algorithms, such as first-come, first-served (FCFS), shortest-job-first (SJF), and Round Robin (RR), have become inefficient and fail to efficiently reduce waiting times or adapt to changes in the system. To address these issues, we introduce the Adaptive Mountain Gazelle Optimizer (AMGO), a modified version of the Mountain Gazelle Optimizer. It has the ability to dynamically balance exploration and exploitation, assigning tasks according to expected processing times, priorities, and deadlines. Its performance was compared with metaheuristic algorithms such as PSO, GWO, and MGO, as well as traditional algorithms. The results showed that the IMGO reduced waiting time by 30.6%, improved makespan completion time by 9.9%, and reduced total cost by 23.1% when examined on 150 tasks distributed across four cores over 100 iterations. These results demonstrate how IMGO improves real-time task scheduling performance. It has become a contemporary alternative to traditional methods in advanced computing systems for its ability to achieve a balance between exploration and exploitation.

Keywords: *GWO, Mountain Gazelle Optimizer, Optimization algorithms, PSO, Task scheduling*

1 INTRODUCTION

Modern computing systems can analyze massive amounts of data at lightning speed due to advances in multi-core processor technology, cloud computing, and smart devices. Task scheduling is a fundamental process that directly impacts system efficiency and overall performance, which pushes researchers to focus on improving task scheduling within CPUs [1].

Traditional algorithms, such as first-come, first-served (FCFS), shortest-job-first (SJF), and round-robin (RR), are not effective in modern systems. Due to the complexity of systems and the diversity of workloads, they can not adapt to dynamic changes in execution environments, resulting in poor resource allocation, long queue times, and unsatisfactory performance for users [2].

Biologically inspired algorithms, such as the particle swarm algorithm (PSO), the gray wolf algorithm (GWO), the ant colony algorithm (ACO), and the genetic algorithm (GA), have gained popularity due to their ability to handle multidimensional data and complex dynamic processing environments. They can balance exploration and exploitation by mimicking the behavior of a gazelle in rugged terrain. The mountain gazelle optimizer (MGO) has become a promising tool for solving complex optimization problems [3].

We propose an improved version of the MGO algorithm called IMGO (Mountain Gazelle Adaptive Optimizer). This version uses dynamic parameters to change the search mechanism based on the progress of the iterations. By employing a scheduling method that varies based on the number of tasks at any given time, this algorithm achieved reduced waiting times, reduced overall execution

time, and maximized utilization of available cores.

This paper presents an adaptive scheduling model that improves performance in multi-core environments and addresses the limitations of traditional algorithms. This model is suitable for cloud computing applications and systems that require low latency and high resource utilization.

The remainder of this paper is structured as follows: Section 2 provides a comprehensive review of the literature and related scheduling techniques. Section 3 defines the methodology, including the design of the algorithm and the system model. Section 4 presents the experimental results and performance analysis. Section 5 concludes the study with the main findings and future research directions.

2 LITERATURE REVIEW

Task scheduling in multiprocessing systems is a fundamental challenge facing researchers in modern computing, especially with the increase in workloads and the complexity of operating environments. Many studies have focused on the use of nature-inspired optimization algorithms to reduce waiting time, energy consumption, and overall completion time (makespan).

Kaur A, Chana I used the Whale Optimization Algorithm (WOA) to schedule multi-objective tasks in cloud systems, achieving a 25–30% reduction in completion time and a 15–20% reduction in energy consumption, despite the significant computational overhead that makes it unsuitable for large-scale systems [4].

In contrast, Zafer et al. integrated the MGO algorithm into a hybrid deep learning model and achieved a low waiting time of 24,114 ms, compared to PSO and GWO. However, it shows slow convergence speeds in large and diverse systems [5].

Sashi and colleagues also proposed a median-average round-robin (MARR) algorithm that successfully reduced the waiting time by 15–20%, with challenges in scaling and adapting to dynamic systems [6].

In another study, Kaur et al. evaluated classical algorithms (FCFS, SJF, and RR) in a simulated cloud environment. They showed that these algorithms suffer from up to 40% increase in waiting time due to their inability to adapt to changes [7].

Tian and colleagues employed the MGO algorithm to schedule tasks at electric vehicle charging stations, achieving energy savings exceeding 30% and a 12–15% reduction in completion time. However, the algorithm required a large number of iterations (around 100) to reach optimal solutions, which is unsuitable for real-time applications [8–12].

These studies indicate that algorithms like MGO offer promising results in improving scheduling performance, but they still face challenges with convergence speed, implementation efficiency in real-world environments, and limited dynamic adaptability. As a result, there is a need to design an improved version of MGO that avoids these drawbacks and achieves a better balance of speed and accuracy, which the present research seeks to achieve through the adaptive IMGO algorithm.

3 MOUNTAIN GAZELLE OPTIMIZATION ALGORITHM

MGO uses four key life dimensions of the mountain gazelle as its optimization methodologies, including Territorial Solitary Males, Bachelor Male Herds, Maternity Herds, and Migration to Search for Food. The efficient use of computational resources in scheduling decision-making processes is achieved through these techniques that attempt to balance exploration and exploitation.

At the top of this hierarchy are the Territorial Solitary Males (TSMs), which represent the optimal solutions discovered up to a given point. This phase enables localized exploitation via elite schedule optimization. It depicts top male gazelles monitoring their territory as they improve positioning for enhanced survival results.

The second behavior is Mother Herds (MHs). This step makes scheduling decisions easier by demonstrating how mother herds protect their young. The algorithm's adaptive dynamic ability to changing search environments contributes to generating new and diverse solutions while simultaneously improving existing ones, creating a balance between exploration and exploitation.

The Bachelor Male Herds (BMH) step maintains different scheduling solutions, which leads to a focus on exploring the search space on a large scale. This helps avoid premature convergence. As this phase is composed of more mobile young males, it prevents the algorithm from trapping in local optima and increases the diversity of the solution set.

These mechanisms are complemented by a Migration for Foraging (MSF) phase, which simulates the movement of gazelles across long distances in search of resources. This step helps create a variety of solutions by allowing deer to move around in search of better options, breaking out of a stagnant state, and expanding their search for

Table 1 Literature review

Study & Year	Approach/Algorithm	Scenario/Application	Key Findings	Limitations/Remarks
Kaur A, Chama I (2020)	Whale Optimization Algorithm (WOA)	Multi-objective cloud task scheduling	Reduced makespan by 25-30% and energy consumption by 15-20% compared to conventional algorithms	High computational overhead, inefficient for large-scale systems
Zafir et al.(2023)	Modified Grey Wolf Optimizer (MGO) in hybrid deep learning	Task scheduling in cloud systems	Minimal waiting time (24, 114 ms over 50 iterations), better than PSO (31, 221 ms) and GWO (33, 146 ms)	Convergence speed issues in large, heterogeneous systems
Sakahi et al (2022)	Median-Average Round Robin (MARR)	Task scheduling. cloud	Waiting time reduced by 15-20% compared to FCFS	Scalability limitations in dynamic, real-time workload systems
Kaur et al (2022)	Classical algorithms (FCFS, SF, RR)	Simulated cloud scheduling	FCFS: makespan 700 ms, SJF: 650 ms, RR: 680 ms Slight makespan improvement	Waiting times increased by up to 40% due to inflexibility.
Tian et al. (2024)	MGO for energy based scheduling	EV charging stations	30% energy savings, 1215% makespan reduction	Slow convergence, avg. 100 iterations needed problematic for real-time use

new areas.

By leveraging the exploration and exploitation phases simultaneously through the use of the four behaviors TSM, MH, BH, and MSF, the system achieves optimal results via exploration and exploitation phases simultaneously, which provides high flexibility in addressing complex problems and quickly reaching optimal solutions in multiple environments.

Figure 1 illustrates how each of these four stages contributes to the improvement process [9].

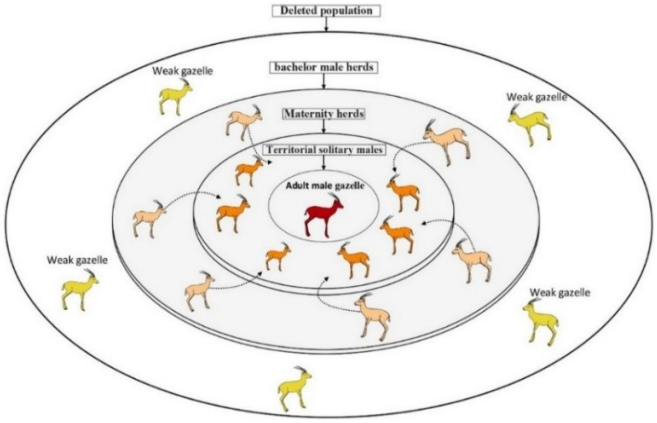

Fig. 1 Hierarchy of Gazelle herds in the inductor algorithm [9]

4 THE PROPOSED ADAPTIVE IMGO – CPU SCHEDULING MODEL

The IMGO (Adaptive Mountain Gazelle Optimizer) algorithm aims to improve task scheduling in multi-core processor environments by reducing waiting and total completion times, while achieving efficient utilization of available cores. This algorithm is an improvement over the original MGO, introducing a dynamic adaptive parameter that helps achieve a balance between exploration and exploitation during iterations.

The first step in the IMGO algorithm is to create a population of size N with random solutions. A solution is defined as the allocation of a number of tasks to available processing resources. Each solution is represented by a vector (X_i), where the j^{th} element indicates the assignment of task j to the core i. The evaluation of each solution is based on two main objectives, the makespan and the wait time for releasing resources. The fitness function can be expressed as:

$$F(X_i) = \alpha \times \text{Makespan}(X_i) + \beta \times \text{WatiingTime}(X_i) \quad (1)$$

Where: X_i is the assignment of tasks to the available cores. α and β are the weighting coefficients that determine the relative importance of Makespan and Waiting Time, respectively. These coefficients were set to 1 to maintain a balance between Exploration and exploitation.

In contrast to the original MGO algorithm, the proposed IMGO adds an adaptation factor (t) to one of the four behaviors to adjust exploration and exploitation at each time step t, and its value ranges from 0 to 1. The algorithm initiates the exploration phase in early iterations to allow searching for potential solutions in the large solution space. Towards the later stages, the adaptive factor becomes smaller, and solutions are exploited. The adaptive factor is computed as follows.

$$\text{adaptfactor}(t) = \text{Max}\left(0.1, 1 - \frac{t}{T}\right) \quad (2)$$

where adaptfactor is a function of time. t is the current iteration, and T is the global iteration limit.

The population update of IMGO can be described as follows:

1. TSM (Territorial Solitary Male): Individual local optimization process.

$$TSM = \text{male}_{\text{gazelle}} - |(ri_1 \times BH - ri_2 \times X(t)) \times F| \times Cof_r \quad (3)$$

The $\text{male}_{\text{gazelle}}$ refers to a location vector of the ideal global solution, whereas ri_1 and $ri_2 \in \{1, 2\}$. BH the young male herds factor, Cof_r is a random coefficient vector used to enhance search efficacy, and its values are updated in each iteration.

Table 2 Comparison between IMGO and MGO

Feature	MGO (Original)	IMGO (Improved)
Parameter Adaptation	Fixed throughout the optimization process	Dynamically adjusted each generation based on task characteristics
Convergence Speed	Good, but slows down for high dimensional problems	Faster convergence via improved initialization and update rules
Task Mapping	Assignment based on static task features	Smart mapping based on priorities, deadlines, and processor availability
Comparison with Other Variants	Does not integrate chaotic or spiral variants	Easily extendable to include chaotic or spiral properties
Explorations vs Exploitation	General balance, non-adaptive	Adaptive balance depending on the optimization stage
Complexity	Less complex, suitable for constrained environments	Slightly higher complexity, but fits dynamic systems
Optimization Strategy	Uses exploration and exploitation mechanisms inspired by mountain gazelles' behavior	Adds an adaptive mechanism to dynamically update parameters for a better exploration-exploitation balance

2. MH (Male Herd): Collective improvement process through solution exchange.

$$MH = \left(BH + Cof_{1,r}\right) + \left(ri_3 \times \text{ male }_{\text{gazelle}} - ri_4 \times X_{\text{rand}}\right) \times Cof_{2,r} \quad (4)$$

where BH represents the scale of the male youth impact factor. Cof $f_{1,r}$, Cof $f_{2,r}$ are randomly selected coefficient vectors, ri_3, ri_4 are integers with values of 1 or 2, while X_{rand} denotes the vector position of a randomly selected gazelle from the group.

3. BMH (Bachelor Male Herd): Random exploration of new solutions.

$$BMH = (X(t) - D) + (ri_5 \times \text{ male gazelle } {}^{-} ri_6 \times BH) \times Cof_r \quad (5)$$

D represents a distance value, $X(t)$ the current position, BH the impact factor of young males, BH, the best solution male gazelle, and finally, the coefficients Cof_r, ri_5, ri_6 are randomly selected at each iteration.

4. MSF (Migration Search for Food): Search for solutions by random alterations.

$$MSF = (ub - lb) \times r_7 + lb \quad (6)$$

where ub, lb are random numbers for the upper and lower limits of the domain. r_7 is a random integer ranging from 0 to 1.

Adjustments based on these processes are made to existing solutions, and each new solution undergoes evaluation using the fitness function. After updates are made, solution candidates are ranked according to their fit value in ascending order. The selection of the best solutions occurs in preparation for the next iteration. The optimal solution X^* is determined as follows:

$$X^* = \arg\min\left(\text{Fitness}(X_1), \text{Fitness}(X_2), \ldots \text{Fitness}(X_N)\right) \quad (7)$$

where X^* is the best solution in the current population.

Modification of the best solution involves applying application of an algorithmic step termed local search, which systematically reorders tasks to be done sequentially in the best solution and evaluates the outcome. The evaluation determines whether an improvement in the optimization results occurs post-adjustment. If an improvement is attained, the modification is accepted. Local improvement takes the form:

$$\mathrm{X}^*_{\text{new}} = \text{LocalRefinement } (\mathrm{X}^*) \quad (8)$$

where X^*_{new} is the refined solution.

To prevent the convergence to suboptimal solutions, a random disturbance is applied to modify the best solution X during any iteration as follows:

$$\mathrm{X}^*_{\text{perturbed}} = \mathrm{X}^* + \text{ Randomperturbed } (\mathrm{X}^*) \quad (9)$$

where $X^*_{\text{perturbed}}$ is the solution obtained after applying a random perturbation.

After N iterations, the algorithm terminates, and the best solution found throughout the entire optimization journey is reported. This solution stands as the representative for the task assignments that minimize the weighted cost function. It optimizes the makespan, minimizes waiting time, and maximally utilizes the available resources.

In the final output, tasks are distributed according to a near-optimal allocation to available cores to improve system performance.

IMGO enhances on the MGO algorithm by adding mechanisms to optimize resource utilization, reducing completion and waiting times in parallel computing environments.

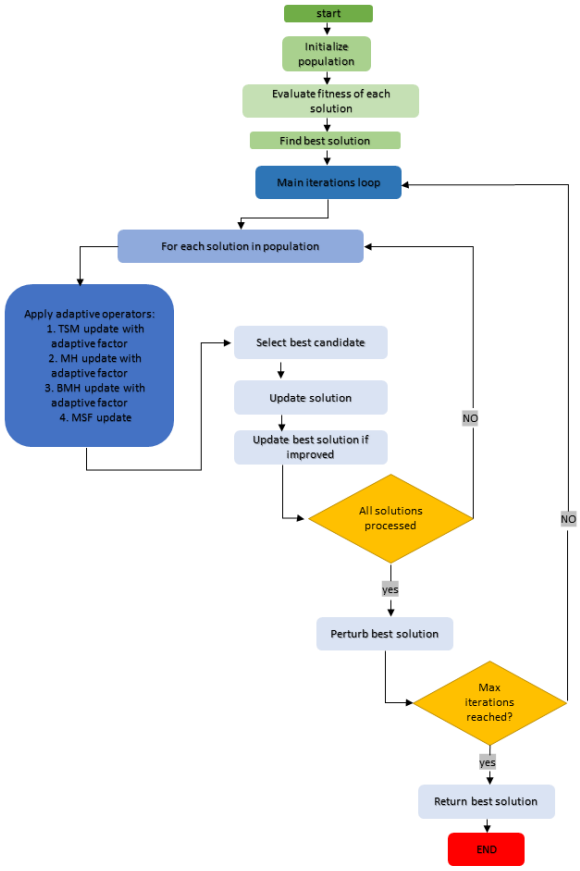

Fig. 2 The procedures of IMGO task scheduling algorithm

The reduced waiting and completion times are the consequences of IMGO's adaptive modification of the original algorithm's four behaviors and its approach to handling complex multitasking workloads by continuously assigning tasks to the most appropriate cores. It also balances global and local search, integrates local modifications, and adapts to changing system conditions, outperforming unresponsive static frameworks in real time. Figure 2 shows the Flowchart of Improves Mountain Gazelle Optimizer (IMGO) Algorithm.

5 EXPERIMENTAL RESULTS AND ANALYSIS

The following section includes experimental results to validate our proposed algorithm in comparison with three alternative algorithms, using a range of metrics, such as waiting time, CPU utilization, total execution time, and cost. It also includes a comprehensive evaluation of the impact of the iteration count on the efficiency of each algorithm and overall scheduling performance.

5.1 Initialization and parameter setting

Each figure is discussed in conjunction with the relevant numerical data for two different iteration counts (50 and 100). By analyzing these figures, we highlight how each algorithm converges, how effectively it schedules tasks, and which method demonstrates superior performance. The subsequent paragraphs offer a concise analysis of every figure, relating observed results to the trends in CPU usage, scheduling efficiency, and total operational cost. This research focuses on enhancing the implementation of the Mountain Gazelle optimization algorithm to improve the adaptive Mountain GO (IMGO) specifically for task scheduling in multi-core systems within the MATLAB environment. The parameters set

of the model include: basic values such as the number of tasks defined as (T a s k s = 150), the number of available cores defined as (C o r e s = 4), and population size defined as (S i z e = 20). In addition, the algorithm includes adaptive dynamic parameters. Table 2 presents a summary of the simulation parameters employed to assess the algorithms.

Table 3 Comparison of Classification Results.

Parameter	Value
numTasks	150
numCores	4
population Size	20
fracHeuristi cMGO	0.4
Alpha + beta	1
maxIterA	50
maxIterB	100
Adaptive Factor	Max(0.1, 1 - Iter/MaxIter)

5.2 Evaluation criteria

The task scheduling problem in multi-core systems is concerned with assigning the execution of a set of tasks, $T = \{t_1, t_2, \ldots, t_n\}$ to a set of available cores $C = \{c_1, c_2, \ldots, c_m\}$. Every task t_i has a set of attributes which include processing time $P(t_i)$, deadline $D(t_i)$, and priority $\Pi(t_i)$. The objective is to determine how tasks are allocated to cores to maximize system performance by minimizing the waiting time and makespan.

The task assignment is assigning each task (t_i) to a core (s_i) in the solution S, where: $S = \{s_1, s_2, \ldots, s_n\}$ and $s_i \in C$.

Waiting Time (WT): It is the total time spent for a task in the queue execution.

$$WT(t_i) = \text{completionTime}(t_i) - P(t_i) - \text{ArrivalTime}(t_i) \quad (10)$$

where completionTime(t_i) is the time at which task t_i finishes, which depends on core s_i to which it is allocated. In this scenario, task scheduling is performed by minimizing these parameters by achieving load balance across the available cores.

Makespan (MS): It is the total time required to complete all tasks. It is defined as the maximum completion time across all tasks.

$$MS(S) = \max_{t_i \in T} (\text{CompletionTime}(t_i)) \quad (11)$$

With the makespan reduction, all assigned tasks are completed in the shortest possible duration, thereby increasing system throughput performance.

CPU Utilization = (Total Execution Time of All Tasks / Total Time Interval) × 100%.
The fitness function $F(S)$ assesses the solution's quality in terms of waiting time and makespan, aiming to minimize both.

CPU Utilization: CPU utilization refers to how effectively processor cores are utilized to achieve a balanced load across all cores to prevent overutilization or underutilization.

In the IMGO-based CPU task scheduling algorithm, CPU utilization is indirectly optimized by minimizing waiting time (WT) and makespan (MS). Task assignments that minimize these metrics result in a more balanced load across cores, preventing idle or overloaded states.

$$CPU_{uti} = \frac{TET}{Tti} \times 100\%$$

where TET is total execution time of all tasks,Tti is total time interval.

While CPU utilization is not explicitly included in the fitness function, the IMGO algorithm naturally maximizes it by minimizing waiting time and makespan, ensuring efficient core utilization and reducing idle time.

5.3 Discussion of results

Figure 3 shows a comparison of the CPU utilization for MGO, IMGO, PSO, and GWO algorithms over an average of 50 iterations. From the figure, it is evident that the PSO and GWO algorithms show greater stability, maintaining a minimum CPU utilization of 30%. Conversely, IMGO shows a higher CPU utilization in the initial iterations, ranging between 35% and 37%. Meanwhile, MGO hovers around 35% in the final iterations. Also, the PSO and GWO algorithms show less dominated update strategies compared to the IMGO algorithm, as shown in the CPU utilization graph, as they conduct less intensive updates, resulting in overall lower CPU usage.

In Figure 4, the makespan (the measured completion time for a specific task) results depict that IMGO has the optimal makespan value with 605 ms, followed by MGO, which recorded 657 ms, PSO, which recorded 662 ms, and GWO recording 700 ms. This indicates that IMGO has a better task allocation strategy as the completion time is lower. Even during the gradual improvements displayed by MGO from approximately 684 to 657, PSO still lags behind IMGO.

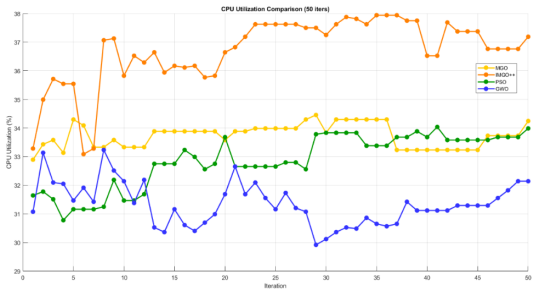

Fig. 3 CPU Utilization Comparison (50 Iterations)

The results also indicate that the GWO exhibits the worst of the four algorithms analyzed, suggesting that its convergence is slower, or the local optima found mid-iteration are fundamentally suboptimal.

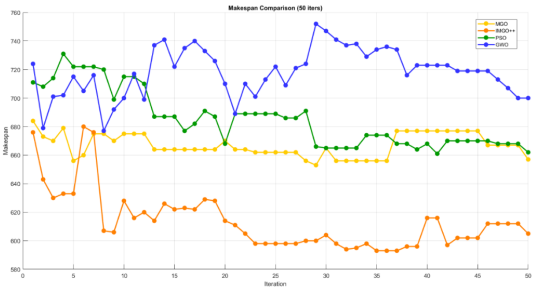

Fig. 4 Makespan Comparison (50 Iterations)

Figure 5 illustrates waiting times (the duration for tasks waiting to be processed), showing that IMGO recorded the lowest total waiting time with 24,114 ms, followed by MGO at 27,620 ms, PSO at 31,221 ms, and GWO at 33,146 ms. The data demonstrates that IMGO has outperformed all competing algorithms by minimizing the queuing latency compared to the standard MGO, PSO, and GWO variants.

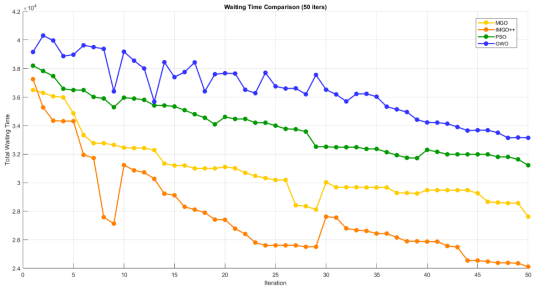

Fig. 5 Waiting Time Comparison (50 Iterations)

Figure 6 illustrates the cost for various algorithms over 50 runs. IMGO stands out for achieving an optimal balance within the weighted cost function due to its superior optimization of makespan and wait time. MGO reinforces IMGO's performance with consistent results, while PSO, despite the reasonable performance, fails to multitask efficiently. GWO incurs the highest cost due to insufficient optimization of makespan and wait time, demonstrating poor performance. These findings support the exceptional IMGO's capability in scheduling tasks where both parameters must be optimized simultaneously.

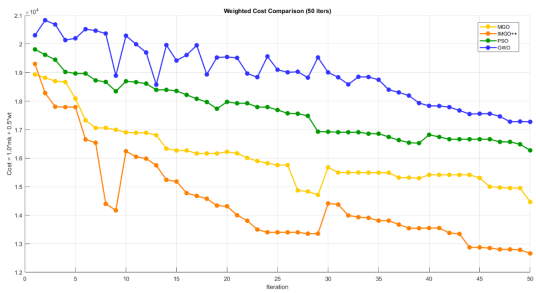

Fig. 6 Weighted Cost Comparison (50 Iterations)

Figure 7 illustrates the trends of Central Processing Unit (CPU) utilization across different task scheduling algorithms over 100 iterations. At the start, the GWO and PSO algorithms operate at relatively low levels between 31-32%. Meanwhile, the utilization for MGO and IMGO averages between 30% and 35% during the initial stages of optimization. By the 100th iteration, IMGO exhibits periodic peaks above 30% while the other algorithms demonstrate a reduction in peak utilization with more stable usage patterns. These findings suggest that IMGO achieves greater workload redundancy due to more sophisticated search mechanisms when compared to GWO and PSO, which employ simpler, less resource-intensive updates. These findings corroborate the superior optimization performance of IMGO presented in Figure 4, which translates to a reduction in operational costs.

Figure 8 illustrates the convergence of makespan for the task scheduling algorithms over 100 iterations. The IMGO algorithm, however, achieved the optimal makespan value at the end of the optimization process ($mk = 598$), outperforming the MGO ($mk \approx 667$), PSO ($mk \approx 633$), and GWO ($mk \approx 636$) algorithms.

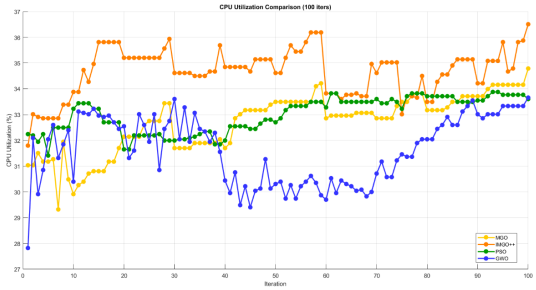

Fig. 7 CPU Utilization Comparison (100 Iterations)

Although all algorithms attempt to reduce makespan as the number of tasks increases, the PSO and GWO methodologies show more divergence towards the convergence point. This illustrates the enhanced optimization capability of the improved IMGO. The exhibited performance gap is due to the employment of the hybrid search strategies in IMGO, which allows more aggressive comparisons in reducing makespan time than the others. These results illustrate the IMGO algorithm's capabilities to adapt to complex multidimensional scheduling problems.

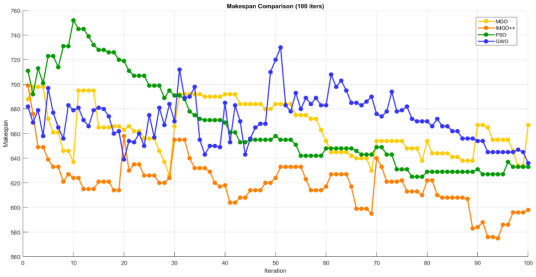

Fig. 8 Makespan Comparison (100 Iterations)

Figure 9 demonstrates the reduction of waiting time for task scheduling algorithms over 100 iterations. We note that IMGO has the lowest wait time ($wt \approx 21,747$), outperforming PSO ($wt \approx 25,190$), MGO ($wt \approx 28,483$), and GWO ($wt \approx 29,463$). The waiting time curve of IMGO shows a steeper decline and faster convergence, which indicates better improvement for load-balanced tasks. On the contrary, PSO and GWO show slow optimization strategies with high wait times and ineffective optimization costs. These findings augment the capability of IMGO in optimizing task scheduling and resource allocation through advanced search strategies despite extended execution durations.

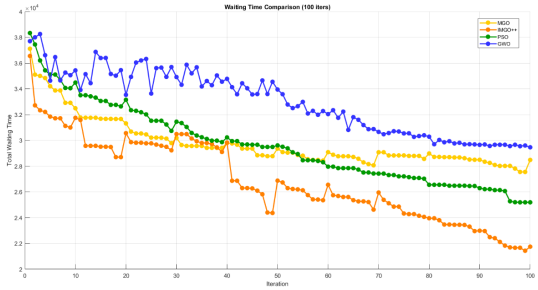

Fig. 9 Waiting Time Comparison (100 Iterations)

Figure 10 illustrates the convergence of the cost function (α Fit) over 100 iterations of different task scheduling algorithms. The IMGO algorithm demonstrated optimal performance by achieving the lowest costs (11,471.50) when optimizing the weighted makespan and waiting times, followed by PSO with approximately 13,228, MGO 14,908.50, and GWO 15,367.50, with progressively poorer cost outcomes. These results indicate that a balance among different objectives was less effectively achieved by the baseline models.

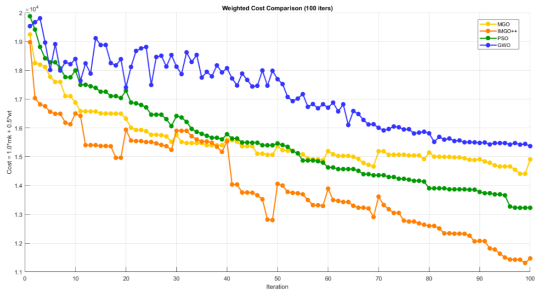

Fig. 10 Weighted Cost Comparison (100 Iterations)

The convergence curve of the IMGO weighted cost function also demonstrates absolute dominance, further evidence of the highly effective hybrid search strategies among conflicting objectives. While average cost stabilization in PSO, MGO, and GWO demonstrated significantly lower performance compared to IMGO, resulting in higher final costs and reduced solution stability. These results collectively support IMGO's multi-target optimization capability and are consistent with the findings presented in Figures 5 and 7, where significant reductions in makespan and waiting time were observed.

6 CONCLUSION

An analysis of four task scheduling algorithms has been conducted, namely IMGO, original MGO, PSO, and GWO, evaluating their CPU utilization, makespan, waiting time, and cost. IMGO achieved the highest performance with a total makespan of 598 Ms, total waiting time of 21,747 MS, and a total weighted cost of 11,471.50 over 100 iterations. The multi-objective hybrid strategies facilitate a balance among all the conflicting objectives, ranked second with 667 MS for the makespan estimate and a cost of about 14,908.50 due to the lack of dynamic optimization. PSO and GWO were further behind with estimated costs of 13,128 and 15,367.50, respectively, with slower convergence due to the imbalance between exploration and exploitation. The results show that IMGO demonstrated the best performance in solving multi-objective scheduling problems. However, building IMGO requires further research to improve the overall efficiency of CPU consumption without compromising performance in resource-constrained systems.

Acknowledgement

N/A

Funding source

No funds received.

Data availability

N/A

DECLARATIONS

Conflict of interest

The authors declare no competing interests.

Consent to publish

N/A

Ethical approval

N/A

REFERENCES

[1] Herrera JL, Scotece D, Galán-Jiménez J, Berrocal J, Di Modica G, Foschini L. MUMIPLOP: Optimal multi-core IoT service placement in fog environments. Computing. 2025;107(4):107. 10.1007/s00607-025-01460-9

[2] Jayamala R, Valarmathi A. FSRmSTS—An Optimize Task Scheduling with a Hybrid Approach: Integrating FCFS, SJF, and RR with Median Standard Time Slice. In: Resource Management in Distributed Systems. vol. 151 of Studies in Big Data. Singapore: Springer; 2024. p. 133-49. 10.1007/978-981-97-2644-8$_7$

[3] Karabiyik MA, Turkoglu B, Asuroglu T. Optimizing Artificial Neural Networks Using Mountain Gazelle Optimizer. IEEE Access. 2025;13:50464-79. 10.1109/ACCESS.2025.3552831

[4] Natesan G, Chokkalingam A. Multi-Objective Task Scheduling Using Hybrid Whale Genetic Optimization Algorithm in Heterogeneous Computing Environment. Wireless Personal Communications. 2020;110(4):1887-913. 10.1007/s11277-019-06817-w

[5] Zafar MH, Mansoor M, Abou Houran M, Khan NM, Khan K, Raza Moosavi SK, et al. Hybrid deep learning model for efficient state of charge estimation of Li-ion batteries in electric vehicles. Energy. 2023;282:128317. 10.1016/j.energy.2023.128317

[6] Sakshi, Sharma C, Sharma S, Kautish S, Alsallami SAM, Khalil EM, et al. A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. Alexandria Engineering Journal. 2022;61(12):10527-38. 10.1016/j.aej.2022.04.006

[7] Kaur R, Laxmi V, Balkrishan. Performance evaluation of task scheduling algorithms in virtual cloud environment to minimize makespan. International Journal of Information Technology. 2022;14:79-93. 10.1007/s41870-021-00753-4

[8] Magdy FEZ, Hasanien HM, Sabry W, Ullah Z, Alkuhayli A, Yakout AH. Mountain Gazelle Algorithm-Based Optimal Control Strategy for Improving LVRT Capability of Grid-Tied Wind Power Stations. IEEE Access. 2023;11:129479-92. 10.1109/ACCESS.2023.3332666

[9] Abdollahzadeh B, Gharehchopogh FS, Khodadadi N, Mirjalili S. Mountain Gazelle Optimizer: A new Nature-inspired Metaheuristic Algorithm for Global Optimization Problems. Advances in Engineering Software. 2022;174:103282. 10.1016/j.advengsoft.2022.103282

[10] Shaker K, Abdullah S, Alqudsi A, Jalab H. Hybridizing Meta-heuristics Approaches for Solving University Course Timetabling Problems. In: Rough Sets

and Knowledge Technology. vol. 8171 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer; 2013. p. 374-84. 10.1007/978-3-642-41299-$8_3$6

[11] Prity FS, Gazi MH, Uddin KMA. A review of task scheduling in cloud computing based on nature-inspired optimization algorithm. Cluster Computing. 2023;26(5):3037-67. 10.1007/s10586-023-04090-y

[12] Ramya K, Ayothi S. Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment. Transactions on Emerging Telecommunications Technologies. 2023;34(5):e4760. 10.1002/ett.4760

How to cite this article

Ahmed AQ, Khamees M, Shaker K. An adaptive mountain gazelle optimizer for minimizing process waiting time in multi-core systems. Journal of University of Anbar for Pure Science. 2026; 20(1):272-281. doi:10.37652/juaps.2025.160843.1400