Nature-inspired Algorithms for Energy-Aware Application Scheduling in Fog Computing Environments

Krisztián Póra[1,2,*], Imre Felde[2] and Attila Csaba Marosi[1]

[1]HUN-REN Institute for Computer Science and Control (HUN-REN SZTAKI), Hungarian Research Network, Kende utca 13-17, H-1111 Budapest, Hungary; krisztian.pora@sztaki.hun-ren.hu, marosi.attila@sztaki.hun-ren.hu

[2]John von Neumann Faculty of Informatics, Óbuda University, Bécsi út 96/b, H-1034 Budapest, Hungary; felde.imre@uni-obuda.hu

* Corresponding author

Abstract: Fog computing provides decentralized resources closer to end-users, reducing latency and improving responsiveness in Internet of Things (IoT) applications. However, energy consumption remains a significant challenge in these diverse environments. This study offers a comparative evaluation of three nature-inspired metaheuristics – Genetic Algorithm (GA), Firework Algorithm (FWA), and Grey Wolf Optimizer (GWO) – for energy-efficient application scheduling in fog infrastructures. We use power-aware fitness functions and extensive parameter tuning to balance exploration and exploitation within a fixed computational budget. With over 100 independent runs, GA achieves the lowest final power consumption, averaging 12.1% less than FWA and 25.4% less than GWO. Although FWA reaches its optimal solution approximately 18% earlier than GA, GA maintains better energy efficiency. GWO converges most slowly and produces less efficient solutions despite sustaining higher population diversity. These results highlight the trade-offs between efficiency, convergence speed and execution time in energy-constrained fog computing deployments.

Keywords: fog computing; nature-inspired algorithms; application scheduling; energy efficiency

1 Introduction

Fog computing has become a crucial paradigm for managing the enormous amount of data produced by the rapidly expanding Internet of Things (IoT). It provides efficient task processing closer to end-users, reduces latency, and enhances resource utilization. This model aims to connect the cloud and edge devices by utilizing decentralized computing resources to improve the performance of various

applications, from smart homes to vital industrial systems. As the demand for energy efficiency grows due to sustainability concerns and operational costs, developing practical scheduling algorithms within fog computing frameworks has become very important. The unique challenges in fog computing – such as fluctuating resource availability, diverse environments, and the need for real-time processing – highlight the necessity for innovative, responsive, and energy-efficient scheduling solutions.

Nature-inspired algorithms have proven to be particularly effective in tackling the complexities of optimization problems, including those encountered in task scheduling within fog computing. Recent studies have highlighted various meta-heuristic strategies that mimic natural processes to enhance task allocation efficiency and minimize energy consumption – for example, Jakwa et al. [1] propose a hybrid meta-heuristics-based task scheduling algorithm that competes favorably against traditional methods, such as modified particle swarm optimization (MPSO). Their results indicate substantial improvements in resource utilization, average response time, and energy consumption, clearly demonstrating the benefits of their approach to resource management for fog nodes.

Moreover, [2] discusses the critical challenges in heuristic task scheduling for IoT applications within fog-cloud computing, outlining the need for algorithms to integrate real-time adaptations and collaborate effectively across devices. This highlights an emerging trend in scheduling where advanced machine learning techniques are proposed to respond to varying workload conditions dynamically, leveraging insights from past performance to optimize future task scheduling decisions.

Additionally, Hussein et al. [3] showcase the potential of ant colony optimization (ACO) in effectively load balancing IoT tasks over fog nodes while minimizing communication costs and response times. The proposed ACO-based scheduling algorithm demonstrates improvements in response times and load balancing compared to traditional methods, reinforcing the applicability of nature-inspired strategies in improving application performance in distributed infrastructures.

While previous studies have examined various nature-inspired algorithms for fog computing, differences in infrastructure models, workloads, and evaluation metrics across publications make it challenging to make direct, quantitative comparisons between methods. Additionally, many works mainly focus on the final solution quality, often neglecting equally important factors such as execution time, convergence speed, and the search dynamics that affect practical use in latency-sensitive environments.

In this study, we offer a controlled, side-by-side comparison of three well-known meta-heuristics – Genetic Algorithm (GA), Firework Algorithm (FWA), and Grey Wolf Optimizer (GWO) – all implemented within the same simulation framework and tested under identical infrastructure and workload conditions. To ensure fairness and practical relevance, each algorithm is tuned and assessed within a fixed

computational budget, reflecting the constraints in time-critical scheduling scenarios where extended optimization runs are not feasible. Our evaluation, supported by statistical significance testing and effect size analysis, measures trade-offs such as GA's consistently higher efficiency across all generations versus FWA's earlier achievement of its final solution, providing useful guidance for choosing algorithms in energy-limited fog computing environments.

The rest of the paper is structured as follows. We present a brief overview of related works in Section 2. In Section 3, we detail the system model and the analyzed nature-inspired algorithms. Section 4 describes the simulation environment of the experiments, the parameter fine-tuning process, and discusses the observed results of the comparative analysis. Finally, section 5 provides conclusions for the work.

2 Related Work

In recent years, the integration of fog computing with practical task scheduling approaches has been a focal point of research, aiming to enhance performance and resource utilization in fog computing environments. Wang et al. [4] present a hybrid heuristic algorithm for task scheduling within smart production lines, leveraging fog computing. Their experimental results show that the proposed strategy performs better than other strategies, highlighting the potential of combined nature-inspired techniques. Similarly, Rafique et al. [5] propose a bio-inspired hybrid algorithm to optimize resource management and task scheduling in fog computing settings. Their simulations in iFogSim [6] demonstrate significant improvements in energy efficiency and scheduling time.

Xu et al. [7] introduce a method that combines laxity and an ant colony system to tackle task scheduling challenges in cloud-fog environments. Their laxity-based priority algorithm organizes tasks based on urgency while considering deadlines, which is crucial for minimizing delays or failures in task execution. On the other hand, Keshri et al. [8] Merge ACO with GWO to effectively address energy consumption and resource wastage in virtual machine placement within cloud data centers.

Domanal et al. [9] further this discussion by developing a hybrid bio-inspired algorithm that integrates features from various methodologies. Their work focuses on reducing operational overhead while maximizing resource allocation efficiency, which is crucial as cloud environments expand.

Adaptive schedulers harness real-time data to respond to workload fluctuations effectively, enhancing overall system performance. For instance, Soula et al. [10] integrate machine learning with bio-inspired heuristics to create an intelligent task allocator that dynamically adjusts to environmental changes, promoting adaptive resource management. Similarly, Nabi et al. [11] propose an adaptive Particle

Swarm Optimization (PSO) approach for task scheduling in cloud computing, minimizing task execution times while improving resource utilization and throughput.

Optimizing the spatial and hierarchical structure of fog environments ensures service proximity and coordination. Talavera et al. [12] employ GAs with hierarchical clustering to refine fog colony layouts, minimizing application response times. Hong et al. [13] extend this by presenting an autonomous evolutionary approach for orchestrating service placement across cloud, fog, and IoT layers, highlighting the value of cross-layer coordination. Similarly, Vakilian et al. [14] explore the role of Artificial Bee Colony algorithms in cooperative load balancing, adding a layer of operational efficiency by promoting inter-node communications that enhance response times and energy costs.

Finally, using the Cuckoo Search Algorithm, Liu et al. [15] tackle multi-objective challenges associated with IoT service placement in fog environments. Their findings reveal that the algorithm successfully navigates trade-offs between competing objectives, thereby aiding the development of robust scheduling strategies.

The reviewed studies show how nature-inspired algorithms are versatile in solving scheduling and resource allocation problems in fog and cloud-fog environments, often leading to significant gains in energy efficiency and performance. However, making direct quantitative comparisons across these studies is challenging due to differences in infrastructure models, workloads, evaluation metrics, and algorithm setups. Additionally, while many studies measure energy consumption, fewer analyze execution time, convergence speed, and the underlying search dynamics – factors that are vital for time-sensitive, resource-limited deployments.

Motivated by these gaps, our work performs a controlled, side-by-side comparison of three representative meta-heuristics – GA, FWA, and GWO – under identical experimental conditions, with tuned parameters and a fixed computational budget, to provide a comprehensive understanding of their trade-offs in energy-aware fog computing.

3 Methodology

This section presents our approach to optimizing task placement in fog computing environments using nature-inspired algorithms. We first describe the system model's components and then detail the three meta-heuristic algorithms implemented: the Genetic Algorithm (GA), the Fireworks Algorithm (FWA) and the Grey Wolf Optimizer (GWO).

3.1 System Model

We model a fog computing infrastructure comprising a source node (e.g., IoT sensor), one central cloud node – acting as destination for the data – and heterogeneous fog nodes dispersed throughout the network. Each fog node i possesses a computational capacity C_i, drawn uniformly at random from a predefined range, and includes both static and dynamic power characteristics: a constant idle draw P_i^{static} and a load-dependent consumption P_i^{dyn} that grows linearly up to a maximum of P_i^{max} at full utilization.

Workloads (formulated as an application consisting of a list of tasks) are represented as directed acyclic graphs, $G = (V, E)$. A single source task is pinned to an edge device and continuously generates data at a fixed rate, while a single sink task resides in the cloud and consumes processed output at its constant rate. Between them, several processing tasks require a certain amount of computation t_i^{req} and may be placed on any fog node whose remaining capacity suffices.

Table 1 summarizes the key system parameters used throughout our study. Numerical values and ranges used for experimentation are detailed within Section 4.

Table 1
Notation in the proposed system model

Symbol	Description
I	Computing infrastructure of interconnected nodes
A	Application made up of tasks to be scheduled
N	Total number of nodes (in the Fog computing layer)
C_i	Computational capacity of node i
U_i	Utilization factor of node i
P_i^{static}	Power consumption of node i in idle state
P_i^{max}	Power consumption of node i at maximum utilization
$P_i^{dyn}(U_i)$	Power consumption of node i at utilization U_i
T_A	Total number of processing tasks in application A
t_i^{req}	Computational capacity requirements of task t

3.2 Nature-inspired Algorithms

We implemented and compared three nature-inspired metaheuristic algorithms for optimizing task placement. Each algorithm minimizes power consumption while respecting computational resource constraints. The algorithms share a common fitness function, which is based on estimating the total power used by application tasks, to avoid the costs of repeatedly allocating and deallocating entire solution populations (which would be even more problematic in real-world scenarios).

We assume that metrics such as static and maximum power consumption, along with actual utilization metrics, are known – typically provided by infrastructure monitoring solution s– and use them to calculate fitness values.

3.2.1 Genetic Algorithm (GA)

The Genetic Algorithm [16] is a population-based metaheuristic optimization algorithm inspired by natural selection and genetic evolution. Algorithm 1 presents the pseudocode for our GA implementation.

The process begins by creating a set of random task-to-node assignments. In each generation, a number of candidates are picked randomly, and superior solutions win "tournaments" to become parents. Pairs of parents then swap genes according to a uniformly random crossover process to produce offspring inheriting a mixture of genes from both selected parents.

To prevent the search from converging too early, each gene in the produced offspring has a small probability (parametrized as the mutation rate) of randomly changing, altering the task assignment of the individual. This mutation step injects fresh variations into the population, promoting diversity. Once enough offspring are created to refill the population, the old generation is replaced and the cycle repeats until the predetermined number of generations is reached. To prevent losing

Algorithm 1 Genetic Algorithm

Input: Population size P, Generations G, Mutation rate M, Tournament size T
Output: Placement solution
Initialize population *pop* with P random solutions
Evaluate fitness of all individuals
Track best solution
for $g = 1$ to G **do**
  new_pop ← ∅
  while |*new_pop*| < P **do**
    parent1 ← TournamentSelect(*pop*)
    parent2 ← TournamentSelect(*pop*)
    child1, child2 ← Crossover(*parent1, parent2*)
    Mutate(*child1, child2*) with M probability per gene
    Add child to *new_pop*
  end while
  Evaluate fitness of *new_pop*
  Replace worst individual with previous best
  Update best solution if improved
  pop ← *new_pop*
end for
return best solution found

previously found promising solutions, elitism was implemented into the algorithm, ensuring that the best solution found so far is kept in the pool of individuals evolving over generations, and that a worse solution will not be returned from the algorithm upon its termination.

3.2.2 Firework Algorithm (FWA)

The Fireworks Algorithm [17] is a population-based meta-heuristic optimization algorithm inspired by the fireworks explosion. The algorithm operates through an iterative process over *G* generations, as outlined in Algorithm 2.

The method initializes *F* fireworks randomly within the search space and sets the amplitude parameter to α. During each generation, the algorithm generates sparks around each firework *f* by creating *S* local sparks within a neighborhood defined by the current amplitude, followed by *R* highly random, so-called Gaussian sparks for enhanced exploration diversity. In our implementation, the amplitude parameter specifies the number of dimensions in which sparks differ from their firework of origin. All generated sparks are collected and evaluated according to the objective function, after which the top *F* sparks are selected to form the next round of fireworks. The algorithm incorporates an elitist strategy by continuously tracking the best solution encountered and replacing the worst spark with the previous best solution. To balance exploration and exploitation, the amplitude parameter is decreased by a decay factor of δ after each generation, progressively narrowing the search radius around promising regions. The algorithm terminates after *G*

Algorithm 2 Firework Algorithm

Input: Fireworks *F*, Sparks *S*, Gaussians *R*, Generations *G*, Amplitude α, Decay δ
Output: Placement solution
Initialize fireworks with *F* random solutions
$amplitude \leftarrow \alpha$
for $g = 1$ to G **do**
  $all\_sparks \leftarrow \emptyset$
  for each firework *f* **do**
    Generate *S* sparks around *f* using $amplitude$
    Generate *R* Gaussian sparks
    Append *f* and its sparks to all_sparks
  end for
  Evaluate fitness of all sparks
  Select top *F* sparks as new fireworks
  Track best solution
  If previous best is better, replace worst spark
  $amplitude \leftarrow amplitude \times \delta$
end for
return best solution found

generations and returns the best solution found during the entire optimization process.

3.2.3 Grey Wolf Optimizer (GWO)

The Grey Wolf Optimizer [18] is a population-based meta-heuristic optimization algorithm inspired by the leadership hierarchy and hunting strategies observed in grey wolf packs. In nature, grey wolves organize themselves into a strict social structure consisting of alpha (leader), beta (second-in-command), delta (subordinates), and omega (followers). The GWO algorithm models this structure to improve candidate solutions within a population of wolves iteratively.

In our implementation (Algorithm 3), a pack of W wolves is initialized with random task-to-node assignments. During each generation, wolves are ranked based on their fitness, and the top three individuals are assigned the roles of alpha (α), beta (β), and delta (δ). These three leaders guide the movement of the remaining wolves in the population (omegas), updating their positions based on a weighted influence of the leaders.

The core mechanism of the GWO algorithm revolves around a simulated encircling and hunting behavior, where each wolf adjusts its position relative to the leaders. The influence coefficients A and C control the balance between exploration

Algorithm 3 Grey Wolf Optimizer

Input: Pack size W, Generations G,
Output: Placement solution
Initialize pack with W random solutions
Evaluate fitness and identify α, β, δ wolves
Track best solution
for $g = 1$ to G **do**
  $a \leftarrow 2(1 - g/G)$ {Linear decrease coefficient}
  for each wolf $w \in W$ **do**
    for each dimension j **do**
      Generate random coefficients A_1, A_2, A_3 and C_1, C_2, C_3
      Calculate distances from α, β, δ
      Calculate position components X_1, X_2, X_3 toward α, β, δ
      $w_j \leftarrow \text{round}\left(\frac{X_1+X_2+X_3}{3}\right) \bmod |N|$ {Ensure valid node index}
    end for
  end for
  Evaluate fitness of all wolves
  If previous best is better, replace worst wolf
  Update α, β, δ
  Track best solution
end for
return best solution found

(searching broadly) and exploitation (refining good solutions). A key parameter a is decreased linearly from 2 to 0 throughout G generations, gradually shifting the search focus from exploration to exploitation.

To maintain diversity and avoid stagnation, each wolf's new position is calculated by averaging its attraction toward α, β, and δ positions, followed by a rounding step to produce valid discrete task placements. The population is updated at each step, and the best solution found so far is tracked and preserved using elitism. This ensures that a superior solution is not lost due to random fluctuations in the search process.

4 Results and Discussion

4.1 Experimental Setup

Our experimental framework is built on LEAF [19], a discrete-event simulator designed explicitly for modeling energy consumption in fog computing environments. LEAF offers strong capabilities for modeling diverse infrastructure with varying computational power and energy use, supports complex application graphs with interdependent tasks and data flows, includes built-in power measurement and monitoring features, and allows efficient execution of large-scale scenarios through discrete-event simulation. The implementation uses LEAF's integrated power monitoring to measure the energy use of tasks and network flows, providing realistic feedback for optimization. Each experiment employed semi-randomized infrastructure and application configurations, with parameters listed in Table 2.

Table 2
Parameter values and ranges used for experimental measurements

Symbol	Value / Range
N	30
C_i	50-150
P_i^{static}	20-40
P_i^{max}	150-250
T_A	100
t_i^{req}	1-15

We assessed each placement algorithm across three main areas. Power consumption measures the total energy used by the application's processing tasks on the underlying infrastructure; by expressing this in watts, we directly evaluate how

energy-efficient a specific algorithm's placement is, which is crucial when deploying services on resource-constrained fog nodes.

On the other hand, execution time measures how long each algorithm takes to make a placement decision; reporting this helps evaluate practical responsiveness, as overly slow placement can negate the benefits of offloading and scheduling in a dynamic network. Additionally, fog computing architectures are often used for transmitting and processing highly diverse workloads (e.g., stream processing), which demand high adaptability.

Since metaheuristic algorithms improve their solutions over multiple generations, just measuring the time they run does not fully assess their performance. The third metric, convergence generation, indicates the exact point during execution when the algorithm reaches a solution quality very close (within 1%) to its final result, showing how quickly it finds high-quality solutions.

These metrics balance the trade-off between energy efficiency and timeliness, ensuring that our evaluation accurately reflects operational costs and real-world practicality. To guarantee statistical validity, we performed 100 independent runs for each algorithm using consistent random seeds across all approaches, allowing for fair comparisons while accounting for the inherent randomness of meta-heuristic methods.

4.2 Parameter Tuning

A uniform computational budget of 100 generations and a population (or pack) size of 50 was enforced across all algorithms to emulate the time constraints of real-world fog computing scenarios, where long optimization runs may be impractical. Within this budget, algorithm-specific parameters were tuned to balance exploration and exploitation without exceeding the computational cap.

For the Genetic Algorithm, we tested tournament sizes from 3 to 9 and mutation rates from 0.01 to 0.1. The heatmap in Figure 1 shows a clear pattern: low mutation rates (especially 0.01) combined with larger tournament sizes (5-7) consistently resulted in the lowest power consumption. This combination enhances selection pressure, allowing high-quality genes to spread quickly while reducing disruptive mutations that could ruin promising partial solutions. Conversely, higher mutation rates increased variability and often lowered overall performance, indicating that in limited budgets, too much exploration can be counterproductive.

The Firework Algorithm tuning analyzed how the fixed group of 50 individuals was divided among core fireworks (F), local sparks (S), and Gaussian sparks (G), along with amplitude (α) and decay factor (δ) settings. Figure 2 shows that most of the population should be assigned to sparks (which focus on intensive searching around good solutions), with only a few core fireworks, plus a small Gaussian component for occasional long jumps, to achieve the best energy efficiency. Starting amplitude

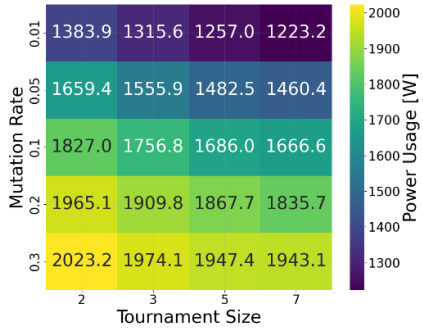

Figure 1

Parameter tuning for the Genetic Algorithm. Each cell in the heatmap represents a specific combination of tournament size (x-axis) and mutation rate (y-axis), with corresponding values indicating the power usage after placement. The color gradient reflects relative performance, where darker shades indicate lower power consumption.

values around 1.0 and a moderate decay rate ($\delta \approx 0.85$) provided enough early exploration while guiding the search in later generations. Faster decays led to premature convergence, whereas slower decays extended exploration without further gains.

The Grey Wolf Optimizer has fewer explicit tuning controls, with behavior mainly driven by the linear decrease of the exploration coefficient a from 2 to 0. We kept this standard schedule, along with the same pack size of 50 for comparison. Although this setup maintains the intended shift from exploration to exploitation, later results indicate that the discrete placement problem may need additional hybridization or leader update improvements to match the efficiency of GA or FWA under strict generation limits. The final parameter settings used in the comparison are summarized in Table 3.

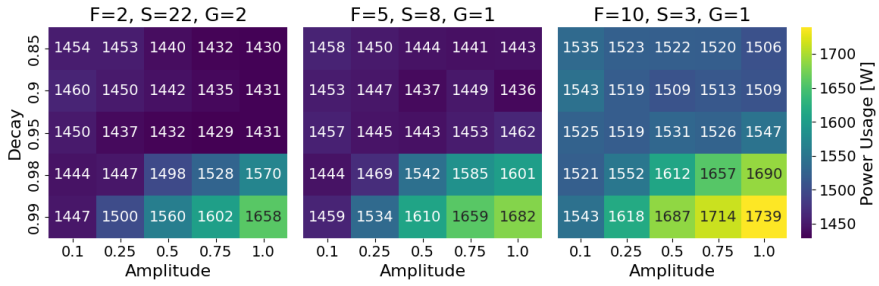

Figure 2

Parameter tuning for the Firework Algorithm. Each subplot corresponds to a different population distribution among Fireworks (F), Sparks (S), and Gaussian Sparks (G), with the total population fixed at 50.

Table 3

Summary of algorithm parameters used for comparison

GA	FWA	GWO
Number of generations: 100		
Population size: 50	F: 2, S: 22, G: 2	Number of wolves: 50
Mutation rate: 0.01	Amplitude: 1	
Tournament size: 7	Decay: 0.85	

4.3 Convergence Dynamics

Figure 3 and Figure 4 together reveal the convergence patterns and the fitness distribution evolution for the populations of each algorithm, enabling a unified view of their search strategies.

The Genetic Algorithm shows the steepest early decline, reducing mean power consumption from about 2200 W to under 1800 W within the first 20 generations. The density heatmap (Figure 4a) reveals a rapid drop in population diversity toward low-power areas, with little variation after generation 30. This early convergence ensures stability – evidenced by GA's narrow standard deviation bands in Figure 3 – but also restricts the discovery of potentially better solutions later, explaining the slower improvements in the final generations.

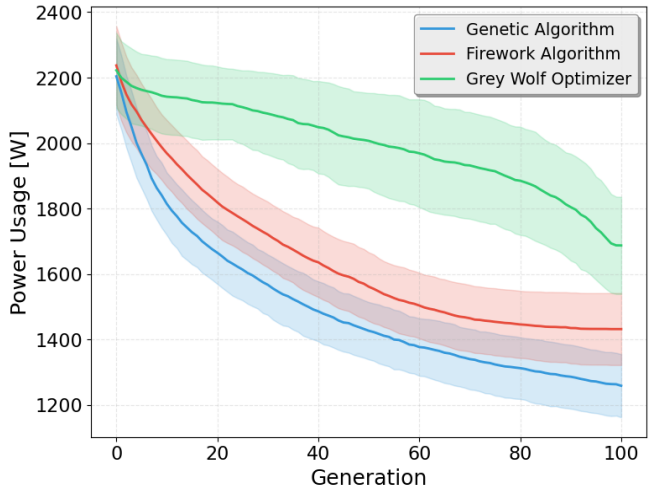

Figure 3

Mean convergence curves for GA, FWA, and GWO over 100 generations. Solid lines represent mean values, while shaded regions indicate ±1 standard deviation.

The Firework Algorithm improves more gradually, with a nearly linear decrease in mean power during the first 40 generations (Figure 3). In Figure 4b, diversity lasts longer as sparks explore local areas while Gaussian sparks introduce occasional high-power outliers. This balance delays the collapse of diversity, allowing the algorithm to keep making steady progress before plateauing. As a result, FWA typically reaches near-final solutions earlier than GA (mean convergence generation ≈ 75 vs. 91), despite slightly higher final power consumption.

The Grey Wolf Optimizer starts at similar initial power levels but exhibits the slowest and most irregular convergence, with ongoing variability across generations (Figure 3). Figure 4c shows that population diversity stays high until late in the process, but without a steady decline in power levels. This indicates that while GWO's leader–follower structure maintains exploration, its exploitation ability is not sufficient in this setup, causing many runs to get stuck in suboptimal regions.

4.4 Comparative Performance and Statistical Validation

Descriptive statistics for all three metrics are reported in Table 4, and pairwise hypothesis tests are summarized in Table 5. GA achieved the lowest mean power consumption (1258.28 W, SD = 97.10 W), followed by FWA (1431.52 W, SD = 110.56 W) and GWO (1687.12 W, SD = 149.17 W). Relative to FWA, GA reduces mean power consumption by 12.1%; relative to GWO, GA reduces mean power consumption by 25.4% (both computed from the reported means). Conversely, GWO consumes approximately 34.1% more power than GA when the increase is expressed relative to GA. These differences are robust: pairwise comparisons of power consumption using paired t-tests yield $p < 0.001$ for all pairs with large effect sizes (Cohen's dz: GA vs FWA = 1.884; GA vs GWO = 3.367; FWA vs GWO =

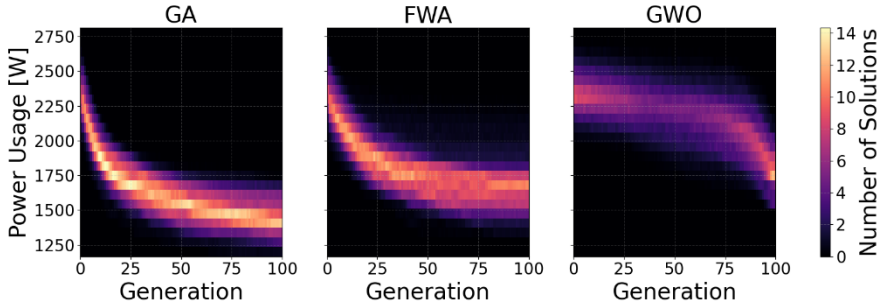

Figure 4

Evolution of population diversity through density heatmaps showing the distribution of solutions across power consumption levels over 100 generations for (a) GA, (b) FWA, and (c) GWO. Color intensity indicates the number of solutions at each power level, revealing distinct exploration-exploitation strategies.

1.786), indicating that the observed differences are both statistically and practically meaningful.

Execution time follows an inverse ranking: FWA is the fastest (mean = 1.01 s), GA is intermediate (mean = 1.07 s), and GWO is the slowest (mean = 1.34 s). FWA is approximately 5.6% faster than GA, while GWO is ~25.2% slower than GA under the chosen implementations. Tests on execution-time within-seed differences (Wilcoxon signed-rank, due to non-normality) returned $p < 0.001$ for all pairwise comparisons with large effect sizes (rank-biserial: GA vs FWA $r = 0.615$; GA vs GWO $r = 0.771$; FWA vs GWO $r = 0.839$), confirming that these differences are significant.

Table 4

Descriptive statistics (100 runs). Mean, 95% confidence interval, standard deviation, minimum, and maximum for power consumption (W), execution time (s), and convergence generation for GA, FWA, and GWO.

Metric	Alg.	Mean	95% CI	Std	Min	Max
Power Usage [W]	GA	1258.28	[1239.01, 1277.54]	97.10	1060.63	1499.33
	FWA	1431.52	[1409.58, 1453.46]	110.56	1160.68	1735.30
	GWO	1687.12	[1657.52, 1716.72]	149.17	1416.76	2068.13
Execution Time [s]	GA	1.07	[1.04, 1.10]	0.15	0.98	1.88
	FWA	1.01	[0.99, 1.03]	0.09	0.94	1.70
	GWO	1.34	[1.30, 1.37]	0.18	1.16	2.23
Convergence Generation	GA	91.09	[89.47, 92.71]	8.16	67.00	100.00
	FWA	74.66	[72.31, 77.01]	11.84	46.00	99.00
	GWO	97.97	[97.54, 98.40]	2.17	85.00	100.00

Convergence generation further differentiates the approaches: FWA typically attains a near-final-quality solution earlier (mean = 74.66 generations) than GA (mean = 91.09 generations) and GWO (mean = 97.97 generations). In relative terms, FWA reaches near-final quality about 18.0% earlier than GA, and GA reaches near-final quality about 7.0% earlier than GWO. These observations are statistically significant (paired t for GA vs FWA: $t = 10.7797$, $p < 0.001$, $dz = 1.078$; Wilcoxon for GA vs GWO: $W = 405.5$, $p < 0.001$, $r = 0.710$; paired t for FWA vs GWO: $t = -19.6359$, $p < 0.001$, $dz = 1.964$).

Taken together, the quantitative results identify a clear trade-off: GA delivers the best final energy efficiency and the most consistent outcomes; FWA is fastest to a near-optimal solution with moderate final efficiency; GWO under the present parametrization is generally slower, more variable, and produces higher-energy placements.

Table 5
Statistical significance tests and effect sizes for pairwise algorithm comparisons (t-p = paired Student's t-test; WSR = Wilcoxon signed-rank; dz = Cohen's dz; r = rank-biserial correlation; all comparisons use n = 100 paired runs.)

Metric	Comparison	Test	Statistic	p-value	Effect Size
Power Usage [W]	GA vs FWA	t-p	t = -11.77	<0.001	dz = 1.88
	GA vs GWO	t-p	t = -24.09	<0.001	dz = 3.37
	FWA vs GWO	t-p	t = -13.77	<0.001	dz = 1.79
Execution Time [s]	GA vs FWA	WSR	W = 737.0	<0.001	r = 0.62
	GA vs GWO	WSR	W = 284.0	<0.001	r = 0.77
	FWA vs GWO	WSR	W = 85.0	<0.001	r = 0.84
Convergence Generation	GA vs FWA	t-p	t = 10.78	<0.001	dz = 1.08
	GA vs GWO	WSR	W= 405.5	<0.001	r = 0.71
	FWA vs GWO	t-p	t = -11.64	<0.001	dz = 1.96

4.5 Discussion

The Genetic Algorithm's superior energy efficiency (average 1258.28W, Table 4) results from its effective combination of evolutionary operators. The tournament selection with a size of 7 creates strong selection pressure, quickly eliminating poor solutions while maintaining diversity through stochastic selection. With seven individuals competing in each tournament, the chance of selecting suboptimal solutions as parents is minimized, guiding the population toward high-quality regions of the search space. The low mutation rate of 0.01 prevents disrupting good solutions while allowing escape from local optima. Most importantly, elitism ensures steady improvement by preserving the best solution found – a crucial feature in discrete optimization where genetic operators might otherwise destroy optimal task assignments. The narrow confidence interval [1239.01, 1277.54] and a coefficient of variation of 7.68% confirm that these mechanisms work reliably across different initial conditions.

The Firework Algorithm achieves the fastest execution time (average 1.01 s, Table 4) through its efficient exploration strategy. The spark generation mechanism evaluates multiple candidate solutions around each firework location in a structured way, requiring fewer fitness evaluations to reach similar solution quality. The 5.6% speed advantage over GA likely comes from the simpler spark generation operations compared to GA's tournament selection and crossover procedures. The smooth convergence seen in Figure 3 results from the amplitude decay mechanism, which provides a clear transition from exploration to exploitation. The configuration with 2 fireworks, 22 regular sparks, and 2 Gaussian sparks (an 11:1 spark-to-firework ratio) shows that intensive local search in a few promising regions outperforms maintaining many independent search origins, reducing computational costs while maintaining solution quality.

The Grey Wolf Optimizer's poor performance across all metrics highlights difficulties in adapting continuous optimization metaphors to discrete problems. The position update mechanism, designed for smooth navigation in continuous spaces, becomes less precise when adapted for valid task-node assignments. The stratified population structure in Figure 5c, while maintaining diversity, shows that many wolves stay far from leaders throughout the optimization process. This explains both the highest average power consumption (1687.12 W) and the most significant standard deviation (148.43 W) observed in Table 4. The late convergence (average 97.97 generations) with a very low variance ($\sigma = 2.16$) suggests that the algorithm consistently needs nearly all available iterations, implying that the linear decrease of the convergence parameter '*a*' might be too cautious for faster adaptation.

Conclusions

This study presented a controlled comparison of three nature-inspired meta-heuristics for energy-aware application scheduling in fog computing:

Genetic Algorithm (GA)

Firework Algorithm (FWA)

Grey Wolf Optimizer (GWO)

The results showed clear trade-offs:

GA consistently delivered the most energy-efficient placements across all generations.

FWA achieved earlier stabilization and the shortest execution times, though with higher final energy consumption.

GWO maintained population diversity but had slower convergence and lower efficiency under the tested conditions.

By analyzing energy efficiency, convergence speed, execution time, and search dynamics together within a fixed computational budget, this work goes beyond the single-metric evaluations that dominate much of the existing literature. The results highlight that algorithm suitability is inherently dependent on context, influenced by the relative importance of rapid scheduling versus maximum efficiency in time-critical, energy-constrained deployments.

Future research will focus on a deeper exploration of the observed search dynamics, including the role of maintaining diversity, balancing exploitation and exploration, and early-stage convergence patterns, to guide the development of hybrid or adaptive strategies. Additionally, expanding the framework to multi-objective optimization will allow for the simultaneous consideration of energy efficiency, quality of service, and other operational goals relevant to fog computing environments.

Acknowledgements

This work was partially funded by the European Commission's GreenDIGIT Horizon Europe project (GA No. 101131207): https://greendigit-project.eu; by the Ministry of Innovation and Technology NRDI Office within the Autonomous Systems National Laboratory Program framework; and the Hungarian project no. TKP2021-NVA-01, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

On behalf of the "ARNL: GPU-enabled cloud-based big data/AI research platform" project we are grateful for the possibility to use HUN-REN Cloud [20] (https://science-cloud.hu/) which helped us achieve the results published in this paper.

References

[1] A. G. Jakwa, A. Y. Gital, S. Boukari, and F. U. Zambuk, 'Performance Evaluation of Hybrid Meta-Heuristics-Based Task Scheduling Algorithm for Energy Efficiency in Fog Computing':, *Int. J. Cloud Appl. Comput.*, Vol. 13, No. 1, pp. 1-16, Jun. 2023, doi: 10.4018/IJCAC.324758

[2] D. Alsadie, 'Advancements in heuristic task scheduling for IoT applications in fog-cloud computing: challenges and prospects', *PeerJ Comput. Sci.*, Vol. 10, p. e2128, Jun. 2024, doi: 10.7717/peerj-cs.2128

[3] M. K. Hussein and M. H. Mousa, 'Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization', *IEEE Access*, Vol. 8, pp. 37191-37201, 2020, doi: 10.1109/ACCESS.2020.2975741

[4] J. Wang and D. Li, 'Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing', *Sensors*, Vol. 19, No. 5, p. 1023, Feb. 2019, doi: 10.3390/s19051023

[5] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, 'A Novel Bio-Inspired Hybrid Algorithm (NBIHA) for Efficient Resource Management in Fog Computing', *IEEE Access*, Vol. 7, pp. 115760-115773, 2019, doi: 10.1109/ACCESS.2019.2924958

[6] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, 'iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments', Jun. 07, 2016, *arXiv*: arXiv:1606.02007, doi: 10.48550/arXiv.1606.02007

[7] J. Xu, Z. Hao, R. Zhang, and X. Sun, 'A Method Based on the Combination of Laxity and Ant Colony System for Cloud-Fog Task Scheduling', *IEEE Access*, Vol. 7, pp. 116218-116226, 2019, doi: 10.1109/ACCESS.2019.2936116

[8] R. Keshri and D. P. Vidyarthi, 'Energy-efficient communication-aware VM placement in cloud datacenter using hybrid ACO–GWO', *Clust. Comput.*, Vol. 27, No. 9, pp. 13047-13074, Dec. 2024, doi: 10.1007/s10586-024-04623-z

[9] S. G. Domanal, R. M. R. Guddeti, and R. Buyya, 'A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment', *IEEE Trans. Serv. Comput.*, Vol. 13, No. 1, pp. 3-15, Jan. 2020, doi: 10.1109/TSC.2017.2679738

[10] M. Soula, A. Karanika, K. Kolomvatsos, C. Anagnostopoulos, and G. Stamoulis, 'Intelligent tasks allocation at the edge based on machine learning and bio-inspired algorithms', *Evol. Syst.*, Vol. 13, No. 2, pp. 221-242, Apr. 2022, doi: 10.1007/s12530-021-09379-0

[11] A. Naouri, N. A. Nouri, A. Khelloufi, A. B. Sada, H. Ning, and S. Dhelim, 'Efficient fog node placement using nature-inspired metaheuristic for IoT applications', *Clust. Comput.*, Vol. 27, No. 6, pp. 8225-8241, Sep. 2024, doi: 10.1007/s10586-024-04409-3

[12] F. Talavera, I. Lera, C. Juiz, and C. Guerrero, 'Optimizing fog colony layout and service placement through genetic algorithms and hierarchical clustering', *Expert Syst. Appl.*, Vol. 254, p. 124372, Nov. 2024, doi: 10.1016/j.eswa.2024.124372

[13] X. Hong, J. Zhang, Y. Shao, and Y. Alizadeh, 'An Autonomous Evolutionary Approach to Planning the IoT Services Placement in the Cloud-Fog-IoT Ecosystem', *J. Grid Comput.*, Vol. 20, No. 3, p. 32, Sep. 2022, doi: 10.1007/s10723-022-09622-1

[14] S. Vakilian, S. V. Moravvej, and A. Fanian, 'Using the Artificial Bee Colony (ABC) Algorithm in Collaboration with the Fog Nodes in the Internet of Things Three-layer Architecture', in *2021 29th Iranian Conference on Electrical Engineering (ICEE)*, Tehran, Iran, Islamic Republic of: IEEE, May 2021, pp. 509-513, doi: 10.1109/ICEE52715.2021.9544399

[15] C. Liu, J. Wang, L. Zhou, and A. Rezaeipanah, 'Solving the Multi-Objective Problem of IoT Service Placement in Fog Computing Using Cuckoo Search Algorithm', *Neural Process. Lett.*, Vol. 54, No. 3, pp. 1823-1854, Jun. 2022, doi: 10.1007/s11063-021-10708-2

[16] D. Whitley, 'A genetic algorithm tutorial', *Stat. Comput.*, Vol. 4, No. 2, Jun. 1994, doi: 10.1007/BF00175354

[17] Y. Tan and Y. Zhu, 'Fireworks Algorithm for Optimization', in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and K. C. Tan, Eds., Berlin, Heidelberg: Springer, 2010, pp. 355-364, doi: 10.1007/978-3-642-13495-1_44

[18] S. Mirjalili, S. M. Mirjalili, and A. Lewis, 'Grey Wolf Optimizer', *Adv. Eng. Softw.*, Vol. 69, pp. 46-61, Mar. 2014, doi: 10.1016/j.advengsoft.2013.12.007

[19] P. Wiesner and L. Thamsen, 'LEAF: Simulating Large Energy-Aware Fog Computing Environments', in *2021 IEEE 5th International Conference on Fog and Edge Computing (ICFEC)*, Melbourne, Australia: IEEE, May 2021, pp. 29-36, doi: 10.1109/ICFEC51620.2021.00012

[20] M. Héder *et al.*, 'The Past, Present and Future of the ELKH Cloud', *Inf. Társad.*, Vol. 22, No. 2, p. 128, Aug. 2022, doi: 10.22503/inftars.XXII.2022.2.8