Hybrid Bacterial Colony Optimization and Particle Swarm Optimization for Load Balancing in Fog Computing

MSc Research Project
Cloud Computing

Aryan Kumar
Student ID: 23310618

School of Computing
National College of Ireland

Supervisor: Prof. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Kumar
Student ID:	23310618
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Prof. Punit Gupta
Submission Due Date:	15/09/2025
Project Title:	Hybrid Bacterial Colony Optimization and Particle Swarm Optimization for Load Balancing in Fog Computing
Word Count:	5985
Page Count:	20

Signature:	Aryan Kumar
Date:	13th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Hybrid Bacterial Colony Optimization and Particle Swarm Optimization for Load Balancing in Fog Computing

Aryan Kumar
23310618

Abstract

Fog computing minimizes latency and bandwidth consumption by processing data near the source, but it has the challenge of workload balancing across the dynamic and resource limited fog nodes. Uneven task assignment can result in bottlenecks, idle resources, and lowered Quality of Service (QoS) standards. In this work, we introduce a hybrid metaheuristic load balancing algorithm for fog computing by combining the Bacterial Colony Optimization (BCO) and the Particle Swarm Optimization (PSO) techniques to enhance load balancing. BCO has good solution space exploration capabilities, whereas PSO has the advantage of quick convergence; the hybrid would take advantage from both to achieve reduced makespan with better VM usage. The proposed algorithm has been developed in the Python language, with original BCO and hybrid modules, along with a standard PSO executable. The program has been tested on a synthetic task–VM generation with size ranges from 100 to 10,000 tasks, with identical experiment settings. The results indicate the hybrid BCO–PSO to exhibit reduced makespan with increased VM utilization compared to the individual BCO, PSO, as well as the Adaptive Inertia Weight Particle Swarm Optimization (AIW–PSO) algorithms for the majority of test scenarios, with faster convergence for high workload scenarios. These experimental results indicate that the proposed hybrid algorithm can be an effective, adaptive solution for real-time task allocation in simulated fog-computing scenarios for heterogeneous fog networks.

1 Introduction

Fog computing is the decentralised approach that extends the capabilities of cloud computing through bringing the computation task, storage of data, and network services closer to the source of the data Sulimani et al. (2024). Unlike the traditional centralised cloud approach this model processes data locally rather than transferring all information to the cloud. This helps to reduce the latency and lowers the bandwidth usage that is very critical factors for Internet of Things (IoT) applications Ogundoyin and Kamil (2023). In a typical three-layer architecture, a large number of interconnected devices are organised into fog nodes and these node performs computational tasks for the connected end devices. These fog nodes does the filtering, aggregate and analyse the data near its source and reduce the workload on cloud resources, improving real-time responsiveness, as illustrated in Figure 1 Sulimani et al. (2024).

Handling and analysing data at the fog layer certainly removes the need to send all raw data to distant cloud servers. As a result, fog computing performs efficiently in terms of latency, energy consumption, network traffic, and task distribution. However, the changing workloads and varying fog resources make effective load balancing a continuous challenge.

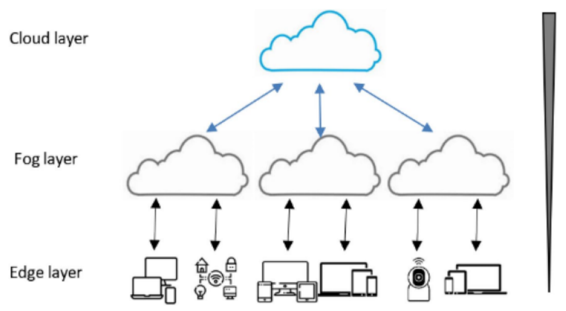


Figure 1: Fog computing architecture Sulimani et al. (2024)

1.1 Research Motivation and Background

Although fog computing certainly decreases the latency for the applications of IoT in the real-time use cases but it is still facing issues with the scheduling task and of load balancing. This still remains a limitation due to the changing workload in the fog environment and the limited resources available at the fog nodes Sulimani et al. (2024).

As some traditional task scheduling methods like Round-Robin were not able to provide good performance with these changing workloads conditions. This caused an issue of long queues of tasks with high energy wastage and poor Quality of Service (QoS) Ogundoyin and Kamil (2023). Because of these, the researchers have choose the metaheuristic algorithms as an alternative method that can be used to design and accommodate these limitations. Among these, Particle Swarm Optimization (PSO) algorithm is well known for its simplicity and fast convergence nature Shaik et al. (2024), but it often suffers from premature convergence problems and poor population diversity in later stages of the search space. Bacterial Colony Optimization (BCO), on the other hand gave a strong exploration nature through the mechanisms like chemotaxis and migration, which is better for more exploration, but this algorithm is likely to converge very slowly Niu and Wang (2012).

To address these disadvantages, some hybrid metaheuristic methods have been explored in different combinations and methods. As Tamilarisi et al. (2021) showed on combining the BCO's exploratory strength with PSO's fast convergence property will improve the accuracy and stability in the tasks. Similarly, hybrid algorithms like PSO–Firefly Ogundoyin and Kamil (2023) and PSO–Simulated Annealing Shaik et al. (2024) have shown promising results in the fog environments, but they still struggles with the dynamic workload variations and maintaining diversity in them.

Motivated by these findings, this study proposes a hybrid BCO-PSO algorithm for task scheduling in a fog computing environment, and it is focused on balancing the exploration behaviour and exploitation nature to of the respective algorithms to improve makespan and uniform load distribution. This approach follows the recent trend of using hybrid swarm intelligence based methods to overcome these optimization challenges in dynamic fog environments. A practical example is a Smart Retail Monitoring system, where dynamic workloads from sensors such as cameras, people counters, smoke detectors, and environmental monitors require balanced scheduling to maintain low latency.

1.2 Research Question

The above research motivation leads to the following research question:

How can a hybrid Bacterial Colony Optimization and Particle Swarm Optimization (BCO-PSO) algorithm improve load balancing in fog computing with respect to minimizing makespan? How does its performance compare to standalone and other hybrid metaheuristic algorithms across different fog node configurations?

This study is organised as follows. Section 2 shows the related work on fog computing challenges and load balancing needs, metaheuristic algorithms, and hybrid methods. Section 3 discusses the methodology that includes research strategy, tools, configurations, and evaluation metrics. Section 4 shows the flow diagram and design of the proposed hybrid BCO-PSO algorithm, and Section 5 covers its implementation and outputs. Section 6 presents the performance evaluation on the makespan, average VM utilization, and execution time, and a discussion. Lastly, Section 7 concludes with key findings and future directions.

2 Related Work

2.1 Fog Computing: Challenges and Load Balancing Needs

Fog computing moves the computation and storage closer to IoT devices through a layered, geographically distributed network of nodes, such as gateways and routers, that reduce the latency and network bottlenecks Mukherjee et al. (2018). Managing these resources is very challenging due to the differences in node capacity, unstable connections, and real-time processing requirements Mouradian et al. (2018). Allocation strategies must consider resource capacity, QoS requirements, and unpredictable workloads. Fog systems must provide low latency with location awareness features and mobility support, which requires a flexible load balancing technique to adapt to shifting workloads and network conditions Naha et al. (2018). Without effective balancing, even a minor delay or imbalance can reduce the entire system's reliability. Many applications have strict deadlines, where poor distribution often causes bottlenecks, idle nodes or missed deadline condition. Smart offloading is important for handling the dynamic workloads efficiently, as poor scheduling leads to latency and energy use in domains such as industrial automation and smart cities Ahmad et al. (2018). Uneven task distribution can also lower service availability and it reflects the need for better adaptive scheduling Abbasi et al. (2018). Effective load balancing is therefore not optional but a fundamental requirement of fog computing, as the uneven allocation causes bottlenecks, idle resources, and degraded QoS.

2.2 Metaheuristic Algorithms in Fog Environments

Metaheuristic algorithms are those algorithms which is inspired by nature and the evolution process and they are widely explored in the fog computing domain for solving the complex optimisation problem exhaustive search Kaur and Aron (2021). They adapt to the changing conditions unlike the traditional methods, and could perform better with changing workloads and flexible network condition. This makes them a better alternative solution for fog environment-related issues.

For example, metaheuristic algorithms like Particle Swarm Optimisation (PSO), Ant Colony Optimisation (ACO) and Genetic Algorithms (GA) are considered good for use cases. ACO algorithm is better managing the dynamic task offloading in the fog environments via reducing delays and improving the performance when compared to other methods Hussein and Mousa (2020). Similarly, PSO has achieved a better response time and higher resource utilisation than traditional approaches in fog–cloud scheduling Ahmad et al. (2018). For energy efficiency, a fuzzy logic–driven metaheuristic approach showed much better workloads and resource availability in real time and reduced both latency and energy consumption Singh et al. (2022). So, these studies show that metaheuristic methods can outperform the existing standalone scheduling techniques and can help fog systems to maintain performance under changing conditions.

2.3 Particle Swarm Optimisation (PSO): Strengths and Drawbacks

The particle swarm optimization (PSO) algorithm is a popular metaheuristic algorithm for load balancing and resource allocation in fog and cloud environments due to its simple and scalable nature and better adaptability Freitas et al. (2020). This is inspired by the flocking behaviour of birds or a school of fish, as PSO represents solutions as particles that update their position depending on their personal best and their neighbourhood experience. This feature provides a much better exploration and exploitation.

PSO algorithm is considered as the alternative for the lower computational overhead that the Genetic Algorithms (GA) or Ant Colony Optimization (ACO) Kaur and Aron (2021). In a fog computing environment, it has reduced the response time and high throughput by the help of Enhanced Dynamic Resource Allocation (EDRA) Baburao et al. (2023). The hybridization method also helped it to enhance its capacity like in PSO–Simulated Annealing for reducing makespan and energy use Shaik et al. (2024), or PSO-Firefly for the better load distribution Ogundoyin and Kamil (2023). The Bacterial Foraging–PSO is also known for its faster convergence in the optimization tasks Reddy and Guduri (2020). This algorithm has also showed adaptability to the changing infrastructure in dynamic load as discussed by Pradhan and Bisoy (2022).

The PSO also have a weakness of a premature convergence problem in a dynamic fog network Freitas et al. (2020), and it is sensitive to the parameter settings such as inertia weight and acceleration coefficients Juneja and Nagar (2016). Effective hybridization and parameter tuning are crucial for its performance in fog load balancing.

2.4 Bacterial Colony Optimization (BCO): Applicability and Gaps

Bacterial Colony Optimization (BCO) was introduced by Niu and Wang (2012) which is inspired by the collective foraging behaviour of *E. coli* bacteria colonies. While Bacterial Foraging Optimization (BFO) which mimic only an individual behaviour of bacteria and lacks inter-agent communication, which could lead to slower convergence, BCO, on the other hand, has social cooperation by implementing chemotaxis, communication, elimination, reproduction and migration with a dynamic step size to balance exploration and exploitation Reddy and Guduri (2020).

It is a swarm-based algorithm with a diversity containing system that makes a BCO more suitable for multimodal and non-linear optimisation in changing environments. While it is largely applied outside fog computing, it has shown potential for having a strong performance in high-dimensional clustering Revathi et al. (2019), and when it is hybridised with PSO, it achieved a better intra-cluster compactness and inter-cluster separation by combining long-term stability with fast convergence Tamilarisi et al. (2021).

Even with these strengths BCO is highly underexplored for fog computing implementation, such as load balancing, energy-efficient scheduling and latency. Due to its dynamic search capabilities and potential for giving better results on hybridization makes it is a promising option for improving load balancing.

2.5 Hybrid Metaheuristics for Load Balancing in Fog

Hybrid metaheuristics approach combines two or more optimisation algorithms to overcome the existing limitations of standalone methods. In fog computing, where workloads changes quickly and resources are very constrained, such combinations can merge the fast convergence of one algorithm with the strong search ability of another for more effective task scheduling.

In fog and IoT–fog environment, a HybOff approach was used that had static–dynamic offloading with the clustering method to reduce decision overhead and gave shorter offloading distance with improved utilisation Sulimani et al. (2024). Harris Hawks–Moth-Flame Optimisation reduced the latency and showed high energy efficiency Saranya and Pabitha (2024), while PSO–Firefly improved the load distribution and gave much lowers makespan. PSO–ACO hybrids method, on the other hand also showed that the cost and scheduling could be improved when compared to their old versions Sabat et al. (2024). The IABC–MBOA hybrids was developed for cloud environments but could be adapted to fog as well Janakiraman and Priya (2021).

Relevant to this study, the Bacterial Foraging–PSO algorithm achieved a faster and more stable convergence Reddy and Guduri (2020) and BCO–PSO hybrids improved the clustering accuracy by combining BCO's search diversity with PSO's convergence speed Tamilarisi et al. (2021). Other PSO-based hybrids such as PSO–Simulated Annealing and NDWPSO (PSO with Differential Evolution and Whale Optimisation) Qiao et al. (2024), addressed the premature convergence problem and improved its quality.

3 Methodology

3.1 Research Strategy

Task scheduling in fog computing is an NP-hard optimisation problem Chandak and Ray (2019) in which the existing methods often fail to adapt to the various workload variations and changing node capabilities. This study proposes a hybrid Bacterial Colony Optimization–Particle Swarm Optimization (BCO-PSO) approach to combine the fast convergence of PSO Freitas et al. (2020) with the stable exploration capability of BCO Niu and Wang (2012) and also considers the Adaptive Inertia Weight PSO (AIW–PSO) for comparison benchmark Nickabadi et al. (2011). The objective is to achieve better load balancing by minimizing the makespan while improving the resource utilisation in fog environments.

3.2 Tools and Environment Setup

All the experiments were implemented in the Python 3.10 using the MEALPY library (v3.0.1) Van Thieu and Mirjalili (2023) for the standard PSO baseline. The BCO and hybrid BCO-PSO algorithms were manually implemented to allow precise control over chemotaxis, reproduction, and swarm updates. The implementation and execution were done on the Google Colab platform, which provided the required computational resources for the algorithms' testing. Data processing, logging and visualisation used NumPy and Matplotlib. Workload datasets were from synthetic JSON files; full details are provided in Section 3.6

3.3 Fitness Function

The objective in this study is *makespan minimisation*, which is a standard metric in the fog computing scheduling that measures the maximum completion time among all the virtual machines (VMs) after the task allocation. A smaller makespan value means a more even distribution of workloads with better utilisation of resources.

Formally, let:

- M = number of VMs,
- T_j = set of tasks assigned to VM j,
- TaskLoad_i = CPU load required by task i,
- VMCapacity_j = processing capacity of VM j.

The makespan is calculated as:

$$\text{Makespan} = \max_{j \in \{1,\ldots,M\}} \sum_{i \in T_j} \frac{\text{TaskLoad}_i}{\text{VMCapacity}_j}$$

where the fraction $\frac{\text{TaskLoad}_i}{\text{VMCapacity}_j}$ represents the execution time of task i on VM j, and the outer max selects the slowest (most loaded) VM.

In the implementation, this equation is realised in the `Fun()` function, which evaluates each candidate solution during optimisation. A small Gaussian perturbation ($\mu = 0$, $\sigma = 0.1$) is applied to each task's VM index before rounding, to promote exploration of neighbouring assignments while preserving the discrete nature of the problem. After the optimisation run, the `Fun1()` function is used for final reporting of the makespan and average VM utilisation based on the best solution found.

3.4 Research Methodology Workflow

The experimental workflow followed in this study is shown in Figure 2:

1. **Dataset Loading:** We load the JSON dataset that contains task CPU load and number of VM required.
2. **Parameter Initialisation:** Set the algorithm parameters like the population size, iteration or the step sizes as required.
3. **Algorithm Execution:** Run all four BCO, PSO, AIW-PSO, and Hybrid BCO-PSO algorithms on the same dataset given.

4. **Fitness Evaluation:** Calculate the makespan for each cases using the Equation 3.3.
5. **Performance Logging:** Save all the best fitness, execution time, and VM utilisation results to a JSON file.
6. **Result Analysis:** Generates convergence curves and performance charts using results.

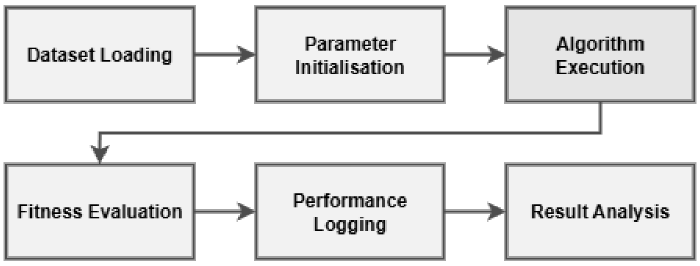


Figure 2: Research Methodology Workflow for Algorithm Evaluation

3.5 Hybrid Algorithm Implementation

The hybrid BCO–PSO algorithm, as described in Section 4.4 was executed alongside with standalone BCO, standard PSO, and the Adaptive Inertia Weight Particle Swarm Optimization (AIW–PSO) to provide a fair performance comparison. The AIW–PSO variant Nickabadi et al. (2011) was selected as a representative adaptive PSO algorithm that introduced a dynamically adjusted inertia weight to improve the balance between exploration and exploitation. For all experiments, the hybrid used the parameter settings listed in Table 1 and operated on the same JSON task–VM datasets. The experiment applied the algorithm to assign tasks to 10 heterogeneous VMs in a simulated fog environment with makespan as the optimisation objective.

Fitness function implementation: The same Gaussian perturbation technique described in Section 3.3 was applied to each task's VM index before rounding to promote exploration while preserving discrete assignments. In addition, the chemotaxis step size C was adjusted adaptively at each iteration, decreasing from $C_{\max}$ to $C_{\min}$ based on the iteration count, enabling broader exploration early on and finer exploitation later. To maintain complete fairness all algorithms were executed under the same parameter settings and dataset configurations with random initialisations for each run.

3.6 Experimental Design

The evaluation process compares the proposed hybrid BCO–PSO against the standalone BCO, standard PSO and AIW–PSO using identical datasets, fitness functions, and execution parameters to maintain fairness. The dataset comprises tasks (with CPU load and execution time attributes) mapped to a fixed set of 10 virtual machines (VMs) with predefined processing capacities. Dataset of various sizes tested were 100, 200,1000, 5000 and 10,000 tasks, which provided scalable analysis. The full simulation hardware and VM configuration is detailed in Section 3.7.

Table 1: Experimental Parameter Settings

Parameter	**Value / Description**
Number of Tasks	100, 200, 1000, 5000, 10,000
Number of VMs	10 (Fixed)
Fitness Function	Makespan minimisation
Execution Platform	Google Colab (Python 3.10)
Library Used	MEALPY v3.0.1
Common Parameters	
Population Size	100 agents
Iterations	50
BCO Parameters	
Step Size Range	$C_{\min} = 0.1$, $C_{\max} = 1.0$
Chemotaxis Power	$n = 2$
Reproduction Interval	20 iterations
PSO Parameters	
Inertia Weight (w)	0.5
Cognitive Coefficient (c_1)	1.5
Social Coefficient (c_2)	1.5
Hybrid BCO–PSO Parameters	
PSO Update Interval	Every 30 iterations
PSO Update Probability	10% per bacterium
PSO Parameters (Hybrid step)	$w = 0.5,\ c_1 = 1.5,\ c_2 = 1.5$

3.7 Simulation Configuration

The experiments were conducted in a simulated heterogeneous environment composed of a cloud data center and three edge server types. Each of the workload scenarios used 10 VMs with tasks assigned in five configurations: 100, 200, 1000, 5000, and 10000 tasks. This range allowed evaluation under light, medium, and heavy loads. The specifications of the simulated servers are provided in Table 2.

Table 2: Configuration of Cloud and Edge Servers

	Cloud Datacentre	**Server Type 1**	**Server Type 2**	**Server Type 3**
Core	200	5	4	8
MIPS	40000	20000	4000	10000
RAM	16000	8000	4000	4000
Storage	1000000	1024000	32000	128000
Bandwidth (MB)	10000	5000	4000	5000

This setup mimics a fog computing deployment where edge nodes differ in processing capacity and memory. It makes sure that algorithms are tested under resource conditions that make the evaluation process more representative.

Metrics. We evaluated the algorithms using three key metrics:

- **Makespan** – The maximum load on any single VM. The lower the number, the better the balance.

- **Average VM Utilisation** – The average (normalised) load across all VMs. Closer to equal means more evenly used.
- **Execution Time** – How long the algorithm takes to run.

Procedure. Each algorithm was executed on all dataset sizes using the same fitness function and parameters. Outputs include final metrics, per-iteration best fitness values, convergence plots, and JSON task–VM assignments were logged for reproducibility and analysis purposes.

3.8 Data Collection and Analysis

Raw performance data from each run, which included the per-iteration best fitness values and final makespan, execution time, and VM utilisation metrics, were automatically saved to a JSON results file after each execution. Each new run appended its metrics to this file, which provides a cumulative storage of results for algorithm dataset combinations. This provides reproducibility and helps in tracking independent runs for each experimental scenario and capturing the natural variability in stochastic optimisation.

The mean, best, and standard deviation were calculated for makespan, execution time, and VM utilisation with all algorithms evaluated using identical datasets, parameters, and stopping criteria. Analysis and visualisation were performed in Python using NumPy for numerical aggregation and Matplotlib for plotting convergence curves, and bar charts were generated separately during the results visualisation stage. Algorithm performance was compared using average metrics, following common practice in fog computing optimisation studies.

4 Design Specification

4.1 System Overview

The proposed system is designed to perform task scheduling in a heterogeneous fog computing environment by using a hybrid BCO–PSO algorithm. It uses the JSON dataset which provides the computational tasks and the capacities of virtual machines (VMs). The scheduling process is focused on minimizing the makespan by distributing the workload evenly across the VMs. The architecture consists of a dataset loader, an optimisation engine implementing the selected scheduling strategy (BCO, PSO, AIW–PSO, and hybrid), and then an evaluation module for calculating the performance metrics.The optimisation engine updates task–VM assignments, while the evaluation module records results for later analysis and comparison.

4.2 Requirements and Assumptions

The system operates under the following:

- Input dataset is in JSON format with task CPU loads, execution times, and a fixed set of 10 VMs.
- Tasks are independent with no precedence constraints or deadlines.
- Candidate solutions are real-valued vectors, discretised to VM indices by rounding; ties are resolved by assigning to the lowest-load VM.

- All tasks must be assigned to exactly one VM.
- Network delays and bandwidth are not modelled; optimisation focuses solely on computation load.
- Parameter settings follow Table 1 in Section 3.

4.3 Architecture Flow Diagram

The architecture provides the complete workflow for executing and evaluating in the simulated fog environment. The process starts with the loading of the data from the JSON dataset, which contains the task CPU requirements and the specifications of the VMs. This data is then passed to the optimisation module, which applies the selected scheduling method. The optimisation loop iteratively updates the candidate task-VM assignments according to the algorithm's search logic and then records the best fitness values at each step.

On completion, the evaluation module then calculates the overall makespan, average VM utilisation, and execution time based on the final assignment. All outputs include the metrics, convergence data, and task-VM mappings, which are stored in the JSON log. This structure ensures that the same dataset with optimisation process and evaluation are applied for comparison across all tested algorithms.

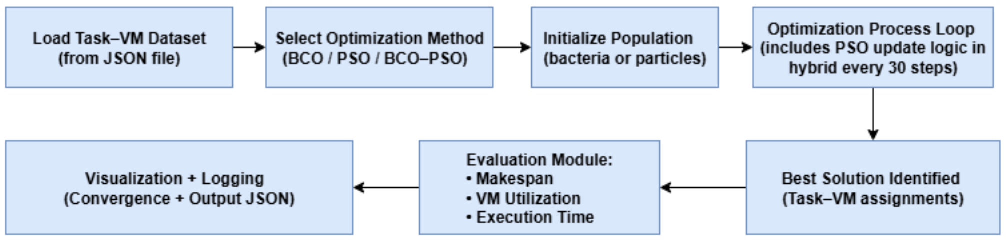


Figure 3: Architecture Flow Diagram for Task Scheduling Algorithms

4.4 Proposed Hybrid BCO–PSO Algorithm

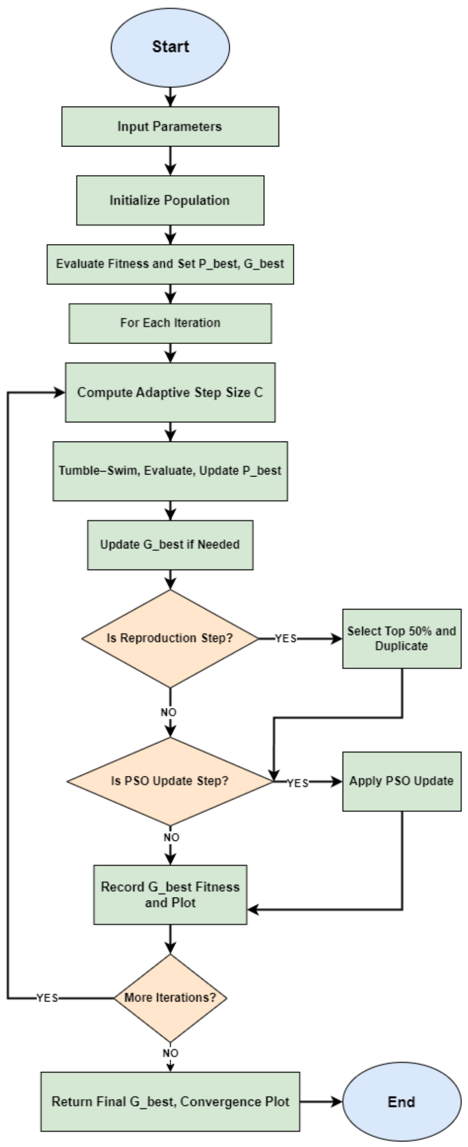
The proposed hybrid BCO-PSO algorithm integrates the adaptive chemotaxis behaviour of Bacterial Colony Optimization (BCO) with the global search capability of Particle Swarm Optimization (PSO) to improve the task scheduling in heterogeneous fog environments. The design is motivated by the complementary strengths of the two algorithms: BCO, which is good at fine grained local exploration and PSO, which accelerates the convergence towards high-quality global optima.

The algorithm begins by randomly initialising a population of bacteria (candidate task-VM assignments) within the VM index bounds. Each bacterium represents a complete schedule and VM assignments. Fitness is then evaluated using the makespan objective, in which the lower values indicating better load balancing.

In each iteration, bacteria perform a chemotaxis process using the adaptive tumble-swim strategy where the step size C decreases over the iterations to balance exploration in early stages and exploitation in later stages. Candidate positions are then updated based on the current orientation, which is modified by a turbulence factor to maintain the diversity and avoid premature convergence.

Reproduction occurs at every 20 iterations, where the top 50% of bacteria (based on fitness) are duplicated to replace the bottom 50%; this process preserves the best-performing schedules while maintaining the population size. Personal best (P_{best}) values are tracked for each bacterium, and the global best (G_{best}) is updated when improvements are found.

Hybridisation with PSO is applied every 30 iterations but only with a small probability (as per parameters in Table 1) to avoid over-dominating the BCO workflow. In this step, velocities are updated according to the standard PSO equation by combining the inertia, cognitive, and social components, and then it is applied to bacterial positions. This targeted PSO injection improves the ability to escape from local optima without sacrificing BCO's adaptability nature. All updated positions are kept within the VM limits and converted to valid VM indices by rounding off, and then evaluated again. The best solution from all iterations is returned with the convergence for analysis. Adapting the continuous algorithms like BCO and PSO into a discrete task-to-VM mapping and then tuning the PSO update interval was the most complex part of this research.



Start
Input Parameters
Initialize Population
Evaluate Fitness and Set P_best, G_best
For Each Iteration
Compute Adaptive Step Size C
Tumble–Swim, Evaluate, Update P_best
Update G_best if Needed
Is Reproduction Step?
YES
Select Top 50% and Duplicate
NO
Is PSO Update Step?
YES
Apply PSO Update
NO
Record G_best Fitness and Plot
YES
More Iterations?
NO
Return Final G_best, Convergence Plot
End

Figure 4: Flowchart of the Proposed Hybrid BCO-PSO Algorithm

Algorithm 1 Hybrid BCO-PSO Algorithm

```
Input:  Task-VM bounds, number of bacteria, max iterations, BCO and PSO
parameters
Initialize population positions and velocities randomly
Evaluate fitness for all individuals
Set personal bests (P_best) and global best (G_best)
for each iteration:
   Compute adaptive step size C
   for each bacterium:
      Move using BCO tumble-swim logic
      Evaluate new fitness
      If improved, update current position and fitness
      If fitness better than P_best, update P_best
   Update G_best if needed
if at reproduction step:
   Select top 50% bacteria and duplicate them
if at PSO update step:
   For each bacterium, with small probability:
      Update velocity using PSO formula
      Update position and re-evaluate fitness
Record G_best fitness at each iteration
Return:  Final G_best, best score, convergence plot
```

4.5 Design Justification

The hybrid design was chosen to overcome the weaknesses of using standalone BCO or PSO for fog-based scheduling. BCO's adaptive tumble-swim nature is good for exploring the nearby solutions in large search spaces Niu and Wang (2012), but it can get stuck without a way to find new promising areas. While PSO can quickly guide the search towards a global best solution Freitas et al. (2020), it sometimes converges too early in the process. Our approach adds PSO updates only at set intervals and with low probability so that BCO can keep its local search strength while PSO helps it to escape from local optima. BCO and PSO were selected because they complement each other's strengths, while GA, ACO, or Firefly were not considered as they have higher computational overhead and poor performance for large-scale fog scheduling. This is why a hybrid approach was chosen instead of relying on BCO or PSO alone.

5 Implementation

5.1 Development of the Simulation Model

This section shows the final implementation of the proposed hybrid BCO-PSO algorithm for load balancing in fog computing environments. The implementation carries the simulation of task scheduling across the virtual machines (VMs) using metaheuristic optimization techniques. All experiments were performed on the Google Colab platform using Python 3.10. The BCO and Hybrid BCO-PSO algorithms were custom-developed in Python, while the standard PSO and AIW-PSO algorithms were executed using the

MEALPY v3.0.1 library.

The final system takes as input a JSON-formatted dataset defining task loads and VM capacities, processes the data using the selected scheduling algorithm, and outputs a task-to-VM assignment optimized to minimize makespan. This implementation then simulates the real fog conditions with varying workloads and resource configurations, tested on synthetic datasets of 100, 200,1,000, 5,000, and 10,000 tasks. The configuration of the simulation environment, tools used, and algorithmic parameters are detailed below, followed by the outputs generated during evaluation.

5.2 Tools and Technologies Used

- **Google Colab:** A cloud-based Jupyter notebook environment was used to run all simulations and visualizations of algorithms.
- **Python 3.10:** The main programming language for algorithm development and simulation execution for the project.
- **MEALPY v3.0.1:** A metaheuristic optimization library that is used to run the PSO and AIW-PSO algorithms for baseline comparison with a hybrid algorithm.
- **Supporting Libraries:** `numpy` (numerical operations), `json` (dataset parsing), `random` (stochastic updates), `time` (execution tracking), and `matplotlib` (visualizations).

5.3 Execution Workflow

In the final stage, the complete simulation environment was set up in such a way that each experiment could run from dataset loading to result visualisation without any manual interruption. This confirms that algorithms were tested under the same conditions and avoid any bias. The workflow was as follows:

1. The simulation notebook in Google Colab firstly loads the specified JSON dataset (e.g., `data_1000_10.json`) from the mounted directory and checks that the task and VM definitions match the expected format.

2. The optimization module receives the dataset and runs the selected algorithm using the same settings for population size, number of iterations, and step size.

3. The loop runs for the set number of iterations. In the Hybrid BCO-PSO case, PSO velocity and position updates are injected at the configured intervals within the BCO chemotaxis–reproduction cycle.

4. At the end of execution, the final task-VM allocation, calculated makespan, average VM utilisation, and iteration-wise best fitness values are saved into a structured JSON results file (`output_final_comparison.json`).

5. Convergence curves were generated automatically by using the stored results while the comparative bar charts were created manually using the exported metrics to provide consistent formatting for all algorithms.

By keeping the workflow under the same conditions across the separate Google Colab notebooks for each algorithm, every run follows the same sequence and settings, which ensures that the evaluation was consistent, reproducible, and free from manual bias.

5.4 Outputs Produced

The simulation produced the following key outputs:

- **Task–VM Assignment:** The computed mapping of tasks to VMs for each algorithm, optimised to minimise makespan while avoiding overload on any single VM.
- **Performance Metrics:** Recorded values for makespan, average VM utilisation (in %), and total execution time for each experimental run.
- **Convergence Curves:** Iteration-wise plots of the best fitness value, which show the speed and stability of convergence.
- **Result Logging:** Structured JSON file (`output_final_comparison.json`) does all the storage of metrics and mappings for the post processing and comparison to take place.

These outputs form the basis of the evaluation presented in the next section, enabling a direct performance comparison of BCO, PSO, AIW-PSO, and the proposed Hybrid BCO-PSO.

6 Evaluation

This section evaluates the proposed Hybrid BCO–PSO algorithm for load balancing in fog computing environments through comparing it with the standalone BCO, standard PSO, and AIW–PSO, which is also included as a known swarm-based scheduling benchmark.The analysis focuses on the three critical performance metrics: makespan, execution time, and average VM utilisation.These metrics are directly responsible for making an impact on the scheduling efficiency, system response time, and resource usage in fog computing. Therefore, it provides a clear basis for assessing whether the proposed hybrid approach achieves its targeted performance improvements.All these experiments were carried out in a Python-based simulation using MEALPY v3.0.1 in Google Colab with task–VM datasets of varying sizes under a simulated fog environment.

6.1 Performance Metric 1: Makespan

Makespan is the total time taken to complete all the tasks in a workload from start to finish. It is an important measure for the load balancing scenarios because it shows how well the tasks are distributed across the available resources. A lower makespan means that tasks are spread out efficiently and avoid delay, and finishing the work in less time. Simply, makespan is like a supermarket with many checkout counters, and it is the time when the last counter finishes serving its customers.

Table 3: Best Makespan results for different algorithms across workloads

Tasks/VMs	Hybrid BCO–PSO	PSO	AIW–PSO	BCO
100/10	4.714	4.714	4.714	5.029
200/10	165.71	177.14	177.14	182.86
1000/10	908.57	914.29	954.29	914.29
5000/10	4616.00	4691.43	6864.00	4668.57
10000/10	9268.57	9296.00	14432.00	9337.14

Interpretation: The proposed Hybrid BCO-PSO consistently gets the lower makespan in almost all the workloads. For smaller workloads (100 and 200 tasks) the performance was similar to PSO. This indicates that the hybrid's advantages are most significant as the task volume and the search space are increased. In larger workloads, improvements were more significant.for example, with 5000 tasks, Hybrid BCO-PSO achieved a makespan of 4616.00 compared to the AIW-PSO's 6864.00, a 31.67% reduction. For 10000 tasks, it reduced makespan by about 0.30% over PSO and 35.8% over AIW-PSO (AIW–PSO originally proposed by Nickabadi et al. (2011)). These improvements come from combining BCO's exploration process which helps to avoid getting stucked on poor solutions too early, and with the PSO's speed-focused updates that quickly improve good solutions and assign tasks more efficiently.

Convergence Behaviour: Figure 5 shows the convergence curve of the Hybrid BCO-PSO for the largest workload (10000 tasks / 10 VMs). The algorithm stabilised near its optimal makespan within 15 iterations, which indicates the fast convergence under high-load conditions, which is particularly beneficial in real-time fog scheduling, where rapid task allocation is critical. This plot serves as a representative example; additional convergence curves for other workloads are provided in the Configuration Manual.

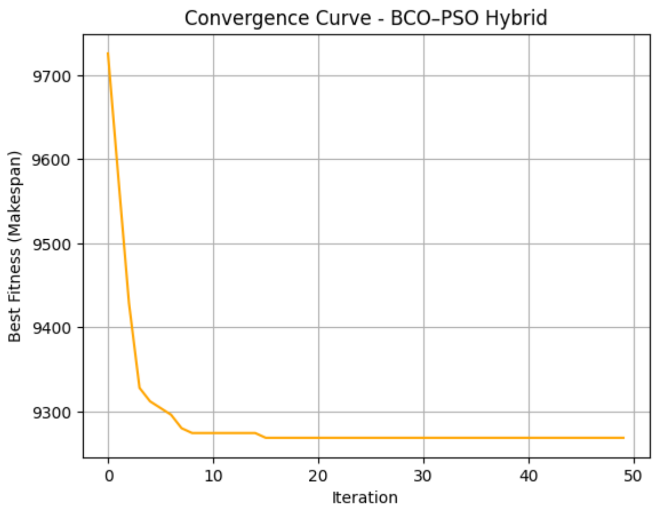


Figure 5: Convergence curve of the Hybrid BCO–PSO for the 10000 tasks / 10 VMs scenario.

6.2 Performance Metric 2: Average VM Utilization

Average VM utilization measures how effectively available VM resources are used during task execution. The higher utilization shows better load balancing.

Table 4: Average VM Utilization (%) for different algorithms across workloads

Tasks/VMs	Hybrid BCO–PSO	AIW–PSO	PSO	BCO
100/10	75.33	75.07	75.33	70.88
200/10	77.93	73.35	73.74	71.31
1000/10	71.40	68.63	71.43	71.09
5000/10	71.17	50.01	69.93	70.30
10000/10	70.90	47.89	69.60	70.26

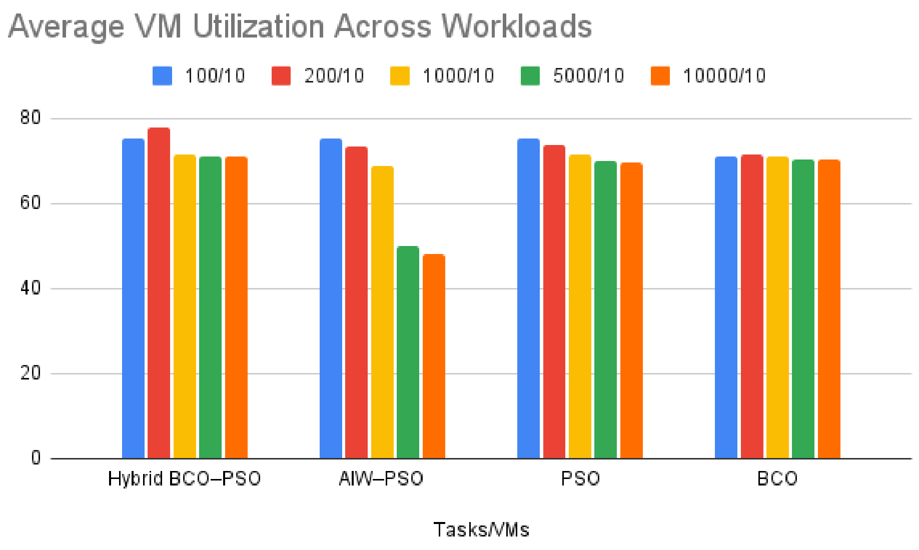


Figure 6: Average VM utilisation across different workloads (100-10000 tasks, 10 VMs). The Hybrid BCO–PSO achieved higher utilisation compared to other algorithms, with a limitation at 1000/10, where PSO was higher. Additional detailed results and convergence curves for all scenarios are provided in the Configuration Manual.

Interpretation: Hybrid BCO–PSO maintained the highest utilization in most cases this peaking at 77.93% for 200 tasks. For smaller workloads (100 and 200 tasks) these utilisation levels were close between Hybrid BCO–PSO and PSO, which indicates balanced resource usage even when demand was low. In large workloads, AIW–PSO's utilization dropped sharply- for example, from 50.01% for 5000 tasks and 47.89% for 10000 tasks. Meanwhile PSO and BCO remained stable but consistently showed below the hybrid model. The hybrid approach's broader exploration phase and refinement step help it to prevent overloading specific VMs, and provide high utilization.

6.3 Performance Metric 3: Execution Time

Execution time is the time taken by the algorithm to complete scheduling, excluding actual task execution on VMs. Lower execution times are beneficial for real-time environments.

Table 5: Execution Time (seconds) for different algorithms across workloads

Tasks/VMs	Hybrid BCO–PSO	AIW–PSO	PSO	BCO
100/10	5.18	8.49	7.36	10.54
200/10	10.55	9.70	9.99	9.05
1000/10	39.28	38.25	37.66	41.13
5000/10	184.83	263.39	187.07	198.10
10000/10	351.80	472.67	512.70	419.64

Interpretation: Execution times varied with workload size in the experiment. Hybrid BCO-PSO was fastest in small workloads (100 tasks) with 5.18 seconds, which benefited from rapid convergence by the help of PSO's velocity updates combined with BCO's initial exploration that avoided wasted iterations. In large workloads (5000 and 10000 tasks), it again outperformed other methods- for example, completing scheduling in 351.80 seconds for 10000 tasks compared to 472.67 seconds for AIW-PSO because the hybrid algorithm's periodic refinement steps reduced the number of iterations needed to stabilise the solution. In medium workloads, BCO was fastest at 200 tasks, while PSO was fastest at 1000 tasks due to its simpler update mechanism and lower computational overhead per iteration. However, the hybrid have a slightly higher cost in these cases, which was offset in larger workloads by better load balancing and faster convergence to high-quality solutions.

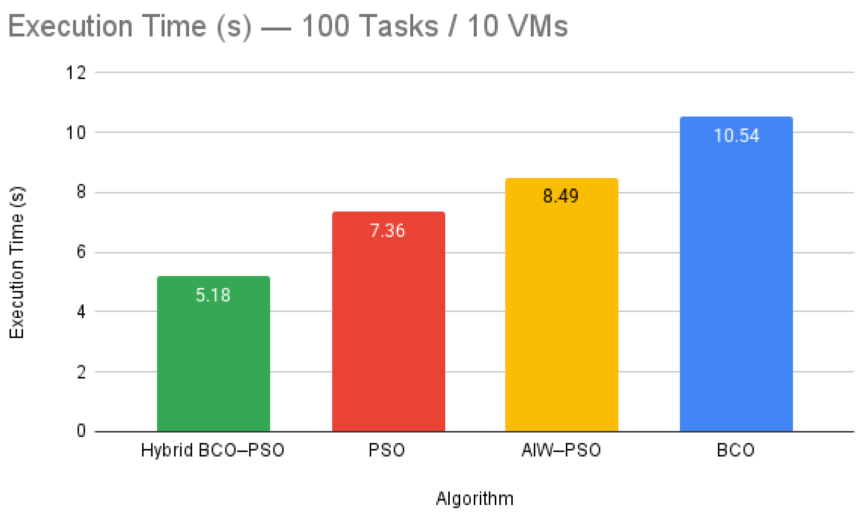


Figure 7: Execution time results for the 100 Tasks / 10 VMs scenario. The Hybrid BCO-PSO achieved the lowest time (5.18 s) compared to PSO, AIW-PSO, and BCO. Additional results and convergence plots are provided in the Configuration Manual.

6.4 Discussion

The experimental results show that the proposed Hybrid BCO-PSO algorithm gave measurable and consistent improvements over the standalone metaheuristics algorithm in the case of fog computing load balancing. In terms of makespan, the hybrid achieved the lowest or near-lowest values across all workloads, with the most significant gains observed in high-load scenarios of 5000 and 10000 tasks. For example, in the 10000-task case, it reduced the makespan by over 35% on compared to the AIW-PSO and also showed better results from PSO by 1.61%. These results are achieved from combining BCO's tumble-swim process that maintains the solution diversity with PSO's velocity-based updates that accelerate the convergence on task-VM assignments are figured out. This allowed the algorithm to adapt effectively to the large search space of high-load cases.

Average VM utilisation results show that the hybrid BCO-PSO has the higher utilisation in most of the scenarios and also peaks at 77.93% for 200 tasks, unlike the case experienced by AIW-PSO under heavy workloads (e.g., 47.89% for 10000 tasks). This means that the algorithm spreads tasks more evenly across different types of VMs due to its exploration method. It first considers the capacity of each VM and then adjusts the task distribution to avoid bottlenecks. In real fog computing systems this helps to keep performance steady and give good Quality of Service (QoS).

Execution time analysis shows that in case of small workloads (100 tasks), the hybrid was the fastest method, completing scheduling in 5.18 seconds by converging rapidly to a high-quality solution. In large workloads also it showed good performance with a 351.80-second runtime for 10000 tasks compared to 472.67 seconds for AIW-PSO and over 500 seconds for standard PSO. These improvements was achieved by the help from earlier solution stabilisation that reduced the unnecessary computation. In medium workloads (200 and 1000 tasks), PSO was marginally faster due to simpler update rules however, this came with slightly higher makespan and lower utilisation. However, the hybrid algorithm also added complexity that may increase the computation overhead when compared to simpler algorithms in medium workloads, and further tuning would be required.

So, the Hybrid BCO-PSO offers a balanced approach that overcomes the stagnation nature of standalone BCO and the premature convergence disadvantage of PSO. The results showed that the algorithm works well in dynamic fog computing environments. It also highlight how combining different metaheuristics can improve performance in several areas. Other relevant metrics, such as energy consumption, network latency, scalability, and fault tolerance, could also be evaluated, but these were outside the scope of this work.

7 Conclusion and Future Work

This study addressed the research question of whether a hybrid metaheuristic can improve load balancing in fog computing by reducing makespan and increasing VM utilisation. The objective was to design, implement, and evaluate a hybrid BCO-PSO algorithm that take advantage of the exploration capability of Bacterial Colony Optimization and the fast convergence nature of Particle Swarm Optimization. Implemented in Python with the custom-developed modules, this algorithm was tested on synthetic task-VM datasets ranging from 100 to 10,000 tasks. The results showed that the hybrid model consistently outperformed the standalone BCO, standard PSO, and AIW–PSO in most scenarios with notable gains under higher workloads. These findings shows that the effectiveness of the approach for heterogeneous fog systems, even its current evaluation is limited to the synthetic datasets and does not model network effects.

Future work will focus on improving the adaptability and real-world application. One direction is for adaptive parameter tuning, where chemotaxis step size and PSO velocity weights would adjust dynamically based on the convergence trends. The method can also be extended to multi-objective optimisation and can jointly minimize makespan and energy consumption to address the observed performance-energy trade-off. Testing under extreme workload patterns, including high task-size variance and bursty arrivals, would further assess the stability in real-time scenarios. Including fault tolerance into the hybrid update process could also improve in case of failures. Finally, migrating to a simulation framework such as CloudSim Plus would allow the modelling of latency, bandwidth, and QoS parameters, and would provide the realistic, network-aware validation and potential commercial deployment in real-world applications. Beyond the scope of this study, the hybrid BCO–PSO could be applied in domains such as smart retail monitoring, healthcare systems, and intelligent traffic management, where dynamic workloads require real-time task scheduling.

References

Abbasi, S. H., Javaid, N., Ashraf, M. H., Mehmood, M., Naeem, M. and Rehman, M. (2018). Load stabilizing in fog computing environment using load balancing algorithm, *International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, Springer, Cham, pp. 737–750.

Ahmad, N., Javaid, N., Mehmood, M., Hayat, M., Ullah, A. W. and Khan, H. A. (2018). Fog-cloud based platform for utilization of resources using load balancing technique, *International Conference on Network-Based Information Systems (NBiS)*, Springer, Cham, pp. 554–567.

Baburao, D., Pavankumar, T. and Prabhu, C. S. R. (2023). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method, *Applied Nanoscience* **13**(2): 1045–1054.

Chandak, A. and Ray, N. K. (2019). A review of load balancing in fog computing, *2019 International Conference on Information Technology (ICIT)*, IEEE, pp. 460–465.

Freitas, D., Lopes, L. G. and Morgado-Dias, F. (2020). Particle swarm optimisation: a historical review up to the current developments, *Entropy* **22**(3): 362.

Hussein, M. K. and Mousa, M. H. (2020). Efficient task offloading for iot-based applications in fog computing using ant colony optimization, *IEEE Access* **8**: 37191–37201.

Janakiraman, S. and Priya, M. D. (2021). Improved artificial bee colony using monarchy butterfly optimization algorithm for load balancing (iabc-mboa-lb) in cloud environments, *Journal of Network and Systems Management* **29**(4): 39.

Juneja, M. and Nagar, S. K. (2016). Particle swarm optimization algorithm and its parameters: A review, *2016 International Conference on Control, Computing, Communication and Materials (ICCCCM)*, IEEE, pp. 1–5.

Kaur, M. and Aron, R. (2021). A systematic study of load balancing approaches in the fog computing environment, *The Journal of Supercomputing* **77**(8): 9202–9247.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J. and Polakos, P. A. (2018). A comprehensive survey on fog computing: State-of-the-art and research challenges, *IEEE Communications Surveys & Tutorials* **20**(1): 416–464.

Mukherjee, M., Shu, L. and Wang, D. (2018). Survey of fog computing: Fundamental, network applications, and research challenges, *IEEE Communications Surveys & Tutorials* **20**(3): 1826–1857.

Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y. and Ranjan, R. (2018). Fog computing: Survey of trends, architectures, requirements, and research directions, *IEEE Access* **6**: 47980–48009.

Nickabadi, A., Ebadzadeh, M. and Safabakhsh, R. (2011). A novel particle swarm optimization algorithm with adaptive inertia weight, *Applied Soft Computing* **11**(4): 3658–3670.

Niu, B. and Wang, H. (2012). Bacterial colony optimization, *Discrete Dynamics in Nature and Society* **2012**(1): 698057.

Ogundoyin, S. O. and Kamil, I. A. (2023). Optimal fog node selection based on hybrid particle swarm optimization and firefly algorithm in dynamic fog computing services, *Engineering Applications of Artificial Intelligence* **121**: 105998.

Pradhan, A. and Bisoy, S. K. (2022). A novel load balancing technique for cloud computing platform based on pso, *Journal of King Saud University - Computer and Information Sciences* **34**(7): 3988–3995.

Qiao, J., Wang, G., Yang, Z., Luo, X., Chen, J., Li, K. and Liu, P. (2024). A hybrid particle swarm optimization algorithm for solving engineering problem, *Scientific Reports* **14**(1): 8357.

Reddy, P. L. and Guduri, Y. (2020). A hybrid bacterial foraging-particle swarm optimization technique for solving optimal reactive power dispatch problem, *International Journal of Applied Power Engineering* **9**(2): 127.

Revathi, J., Eswaramurthy, V. P. and Padmavathi, P. (2019). Bacterial colony optimization for data clustering, *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, IEEE, pp. 1–4.

Sabat, N. R., Sahoo, R. R., Pradhan, M. R. and Acharya, B. (2024). Hybrid technique for optimal task scheduling in cloud computing environments, *TELKOMNIKA (Telecommunication Computing Electronics and Control)* **22**(2): 380–392.

Saranya, M. and Pabitha, P. (2024). Hybrid multi-objective harris-hawks and moth-flame optimization algorithm for efficient task offloading strategy in iot-based fog computing applications, *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, IEEE, pp. 1–6.

Shaik, M. B., Reddy, K. S., Chokkanathan, K., Biabani, S. A. A., Shanmugaraja, P. and Brabin, D. R. D. (2024). A hybrid particle swarm optimization and simulated annealing with load balancing mechanism for resource allocation in fog-cloud environments, *IEEE Access* **12**: 172439–172460.

Singh, S. P., Kumar, R., Sharma, A. and Nayyar, A. (2022). Leveraging energy-efficient load balancing algorithms in fog computing, *Concurrency and Computation: Practice and Experience* **34**(13): e5913.

Sulimani, H., Sulimani, R., Ramezani, F., Naderpour, M., Huo, H., Jan, T. and Prasad, M. (2024). Hyboff: a hybrid offloading approach to improve load balancing in fog environments, *Journal of Cloud Computing* **13**(1): 113.

Tamilarisi, K., Gogulkumar, M. and Velusamy, K. (2021). Data clustering using bacterial colony optimization with particle swarm optimization, *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, IEEE, pp. 1–5.

Van Thieu, N. and Mirjalili, S. (2023). MEALPY: An open-source library for latest meta-heuristic algorithms in Python, *Journal of Systems Architecture* **139**: 102871.