Optimizing Fog Computing Task Scheduling: Selection and Superiority of Differential Evolution Algorithm

MSc Research Project
MSc in Cloud Computing

Joan Bency
Student ID: 21222959

School of Computing
National College of Ireland

Supervisor: Prof. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Joan Bency
Student ID:	21222959
Programme:	MSc in Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Punit Gupta
Submission Due Date:	18/09/2023
Project Title:	Optimizing Fog Computing Task Scheduling: Selection and Superiority of Differential Evolution Algorithm
Word Count:	7209
Page Count:	22

Signature:	
Date:	18th September 2023

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):	

Optimizing Fog Computing Task Scheduling: Selection and Superiority of Differential Evolution Algorithm

Joan Bency
21222959

Abstract

The increase in the number of Internet of Things (IoT) devices raises the amount of data generated every day. Cloud computing provides storage, processing, and analytical capabilities to handle such huge amounts of data. Furthermore, some applications cannot handle the elevated latency and consumption of bandwidth. To solve this, the fog computing paradigm puts cloud services nearer to the network edge. Yet, because fog resources are varied, resource-constrained, and distributed, efficient task scheduling becomes crucial for increasing performance. Response time to application requests, as well as bandwidth and CPU usage, can be decreased with an efficient job scheduling algorithm. This work describes a thorough investigation into task scheduling in fog computing using meta-heuristic techniques. Many evolutionary and swarm-based methodologies were explored and simulated by combining them with the PureEdgeSim simulator, which realistically simulates cloud-fog situations and differential evolution(specifically SADE) emerged as the best option due to its superior performance. Compared to the Bees Algorithm, SADE showed a decrease of 87.9% in failed tasks and compared to BaseGA, it showed a decrease of 15% in execution delay. Also compared to OriginalPSO, SADE exhibited a 5.4% decrease in average bandwidth and a 23.2% decrease in average CPU usage. This research advances the field of efficient fog computing optimisation while emphasising the importance of method selection in tackling real-world difficulties.

1 Introduction

The Internet of Things(IoT) is one of the technologies that has the prospect of providing a great deal of benefits to the world. Many of the devices and gadgets around us are changing in a way that they are able to be connected to the internet and are able to communicate with each other without any human intervention. According to a report by McKinsey Global Institute(1), IoT could have a potential economic impact of \$3.9 trillion to \$11.1 trillion per year by 2025 across several industries. Another report by Accenture(2) found that industrial-level IoT has the possibility to add \$14.2 trillion to the global economy by 2030. As such communication happens the need for manual data entry is reduced as data is collected from the environment using various sensors and scanners and this data is stored and processed automatically. IoT environment produces an unprecedented amount of data that needs to be stored, processed and analyzed all the time. This data is used for various decision-making processes in the immediate and distant future.(3)(4).

Even with all the advantages IoT has, its limitations make it harder to implement it extensively. One major disadvantage was the limited computing and storage power. But that was solved by integrating the Cloud with IoT for storage and computing. In cloud computing, service users have access to efficient and adaptable services. In traditional cloud computing, the two primary players are Cloud Providers and Cloud Users. Provider resources like storage and processing are leased by users and are paid as per their use. A high processing bandwidth is needed to transfer data between the end user and the cloud which can cause delays (4)(5).

Thus, in industries like healthcare, emergency response and other such latency-sensitive ones where speed is an important factor, the cloud can be a liability because of the delay caused while transferring data to and from the cloud. Also, sending so much data to the cloud for storage and processing is inefficient because it would saturate network bandwidth and is not scalable(3)(4).

As a solution to these issues, it was proposed to use computing resources near IoT sensors for local storage and initial data processing calling it edge computing. Edge Computing would reduce network congestion and speed up analysis and decision-making as a result. Yet, edge devices are incapable of managing multiple IoT applications competing for their limited resources. These constraints are overcome by fog computing which quite easily integrates edge devices and cloud resources. By utilising cloud resources and coordinating the use of geographically dispersed edge devices, it prevents resource contention at the edge(4).

Fog Computing is a highly virtualized platform that offers networking, storage, and computing services between end devices and conventional cloud computing data centres, which are typically but not always situated at the edge of the network. Essentially the technology manages IoT data locally by relying on clients or edge devices close to users to handle a sizable amount of storage, communication, control, configuration, and management. The method takes advantage of the close proximity of edge devices to sensors while utilising the flexibility of cloud resources to scale on demand. Applications for data processing or analytics running on distributed clouds and edge devices are included in fog computing. Additionally, it makes it easier to manage and program storage, networking, and computing services between data centres and endpoints(4)(6).

While having all these advantages, it is important to schedule the incoming tasks efficiently in fog computing. While fog computing compensates for the limitations of edge computing by integrating edge devices with the cloud, it is important to have a good task scheduling method so that tasks will be assigned correctly. That is, tasks that can be handled by edge devices need to be assigned to the edge and when the task is too big or complicated and is outside the computing power of an edge device it needs to be assigned to the cloud, or such methods need to be taken. All of these need to be done while keeping in mind several metrics like delay, bandwidth, network usage, energy consumption, etc.(7)

There are different types of task-scheduling algorithms in fog computing. Some of the common types are:

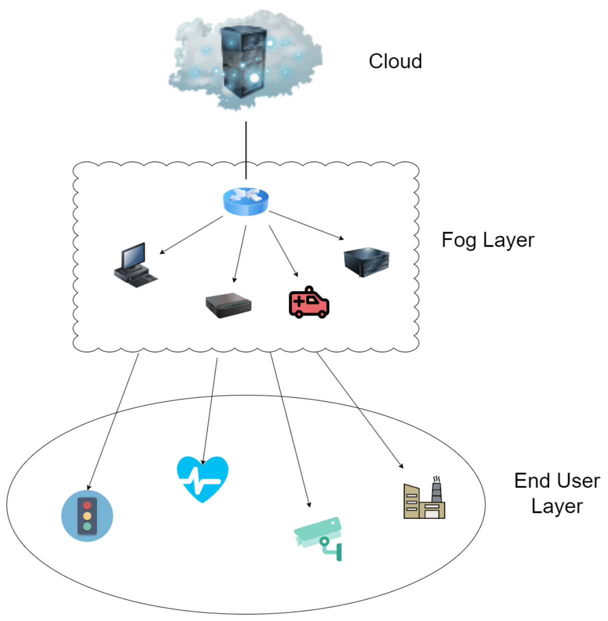


Figure 1: Cloud Fog Environment Architecture

1.1 Heuristic Algorithms

Heuristic Algorithms are simple and fast and the rules they use are predefined. The solutions they give may not always be optimal ones, but they can always handle large and dynamic problems. Some examples of heuristic algorithms are First Come First Serve(FCFS), Shortest Job First(SJF), Round Robin(RR) and Priority-based algorithms(8).

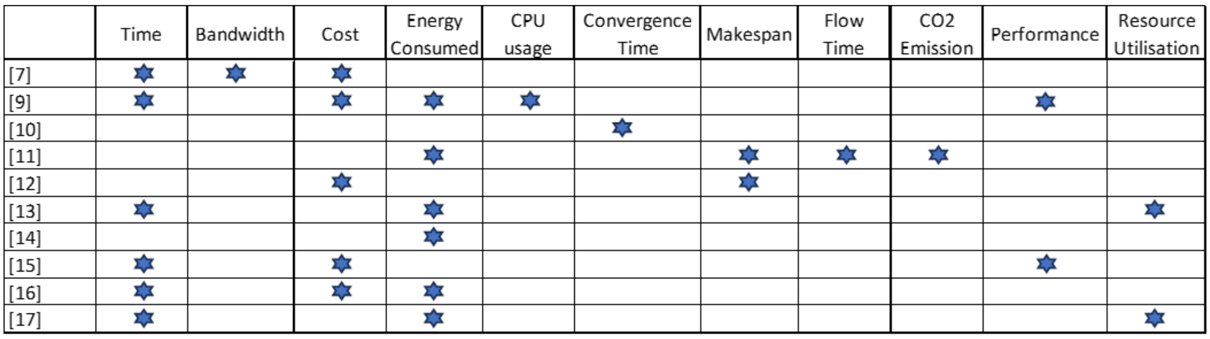
1.2 Meta-heuristic Algorithms

These algorithms are intelligent and advanced and they use evolutionary or stochastic techniques to find near-optimal methods to schedule tasks. These algorithms can handle complex problems but may require more time and computational resources. A few examples are Genetic Algorithms(GA), Ant Colony Optimization(ACO) algorithm, Particle Swarm Optimization(PSO) algorithm, and Differential Evolution(DE) algorithm(8).

In this paper, we will be presenting an efficient scheduling technique for a cloud-fog computing environment. From all experiments, the suggested solution is a Differential Evolution(DE) algorithm. This scheduling approach considers various features like delay, tasks failed, network usage, bandwidth, and CPU usage. Furthermore, energy consumption is also kept in mind. This solution has been utilized to schedule tasks, utilising the PureEdgeSim simulator to confirm the results. The results of the experiment are compared with other advanced evolutionary and swarm-based algorithms.

Fog computing is a paradigm for computing that allows for processing, storage and communication at the network's end. Large-scale applications can be processed and stored using cloud resources with fog computing. In cloud fog architecture, the cloud layer is the top layer, in the middle are the nodes or devices acting as the fog devices, and the bottom layer consists of end users(7). Figure 1 presents the architecture of the Cloud fog environment.

The Cloud Layer is the top layer of the cloud fog environment. This is the layer where storing and processing of data happens over the internet. The cloud provides large-scale



	Time	Bandwidth	Cost	Energy Consumed	CPU usage	Convergence Time	Makespan	Flow Time	CO2 Emission	Performance	Resource Utilisation
[7]	★	★	★								
[9]	★		★	★	★					★	
[10]						★					
[11]				★			★	★	★		
[12]			★				★				
[13]	★			★							★
[14]				★							
[15]	★		★							★	
[16]	★		★	★							
[17]	★			★							★

Figure 2: This is a table comparing all Related Works

and centralized resources like storage and computing power. This layer should be used for applications or tasks that are too large and complex for edge or fog devices. The biggest disadvantage of the cloud layer is the high latency. Some examples where this layer is used are data mining, machine learning, etc(7).

The fog layer is the middle layer and is where cloud resources are spread to the edge of the network, i.e. nearer to the end users and devices. Fog computing offers resources and services that are distributed and in the middle. Applications that need low latency, and high bandwidth, but not high performance should use fog computing. Fog computing is widely used in smart cities, smart healthcare and vehicle networks(7).

The end layer, which contains numerous heterogeneous and mobile devices is the layer closest to the end user. End devices most frequently run on batteries have comparatively low CPU and memory capacities which has a negative impact on battery life. As the processing and storage capacity is limited, data generated by end devices are mostly processed or stored elsewhere. Gadgets like sensors(heart monitors), cell phones, and cameras are some end devices in the Internet of Things(IoT)(7).

The remainder of the paper is arranged as follows: Second Section consists of the related works. Section 3 covers Implementation. Implementation results will be discussed in Section 4 and Section 5 will include the conclusion of the paper.

2 Related Work

The task scheduling and task allocation strategies that are currently in use to boost performance in a cloud fog environment are covered in this section. The scheduling of jobs from fog to the cloud is done by resource optimizations. By cutting down on execution time, reducing network usage, lowering costs, etc., these schedules and allocations are meant to deliver a higher QoS. Here is an overview of a few of these methods for task scheduling.

In (7), Arshed et al. proposed an efficient task-scheduling technique that is mainly for IoT devices. This is a genetic algorithm that reduces reaction time to application requests along with bandwidth and cloud service costs. Execution time is used as a fitness function in the suggested solution to choose an effective module scheduling across the available fog devices and in terms of execution time, financial gain and bandwidth this solution performs way better than baseline algorithms. This solution performs 15-40% better compared to simple algorithms, i.e. it is quicker, consumes less bandwidth and has lower financial costs. But as it says even though it performs better, it is only better than the simpler baseline algorithms.

In (9), Natesha and Guddeti proposed a technique primarily for IoT devices. Here, using docker and containers a two-level resource provisioning fog framework was created, and the service placement problem in the fog computing environment was formulated as a multi-objective optimisation problem to reduce service time, cost, and energy consumption. It makes use of the Elitism-based Genetic Algorithm((EGA). It was tested on devices with 1.4 GHz 64-bit quad-core processors and a docker and container-based fog computing testbed. In terms of service cost, energy consumption, service time, and average CPU utilisation of fog nodes, the proposed method performs better than other service placement strategies taken into consideration for performance evaluation. One of the work's limitations is that the method did not take into account the interconnected IoT applications to evaluate how well the proposed EGA performed on the created fog testbed.

In (10), Hoseiny et al. proposed a task-scheduling algorithm that serves primarily IoT devices. A fundamental problem is how to effectively use the resources of the fog cloud to carry out tasks that are offloaded from IoT devices. Here they proposed a fog-cloud scheduling algorithm called PGA to maximise the multi-objective function, which is a weighted average of total computation time, energy usage, and the proportion of tasks that are completed by the deadline (PDST). The various task requirements as well as the diverse makeup of the fog and cloud nodes were taken into account. To find the best computing node for each task, they suggested a hybrid technique based on task prioritisation and a genetic algorithm. It performs noticeably better than the compared algorithms and offers a decent convergence time.

In (11), Abdel-Basset et al. proposed another algorithm that is primarily used for IoT devices. It is an improved elitism genetic algorithm (IEGA) for Fog Computing (FC) to solve the task scheduling issue and raise the QoS provided to users of IoT devices. The two main phases of IEGA's improvements are the manipulation of the mutation rate and crossover rate, which enable the algorithms to explore the majority of the possible combinations that could result in the near-optimal permutation and to mutate various solutions based on a given probability in order to evade getting stuck in local minima and discover a more acceptable solution. In terms of energy consumption, fitness function, makespan, flow time, carbon dioxide emission rate, and fitness function, it is compared with five recent powerful optimisation algorithms and EGA, and IEGA is superior in all metrics. However, for task sizes greater than 700, the proposed approach outperforms the others in terms of flow time, demonstrating its superiority for large-scale problems.

In (12), Ali et al. suggested an algorithm that minimises both the makespans and total costs in a fog-cloud environment by solving a multi-objective task-scheduling optimisation problem. Then, we propose an optimisation model for the discrete multi-objective task-scheduling problem based on a discrete non-dominated sorting genetic algorithm and automatically assign tasks that should be executed either on fog or cloud nodes. Instead of using continuous operators, which are resource-intensive and incapable of allocating suitable computing nodes, the NSGA-II algorithm is modified to discretize the crossover and mutation evolutionary operators. To improve the execution, communications between the cloud and fog tiers are formulated as a multi-objective function. The suggested model organises the distribution of workloads among various computing resources at the fog and allots computing resources that would run on either fog or cloud nodes. It is contrasted with four peer mechanisms and a continuous NSGA-II (CNSGA-II) algorithm. Results show that by reducing makespan and expenses in fog-cloud environments, the model can achieve dynamic task scheduling. It can make use of the model to distribute batch tasks involving large amounts of data in environments with fog clouds.

In (13), Singhrova proposes an effective resource allocation algorithm in fog computing, which is a hybrid Prioritized Genetic Particle Swarm Optimization (P-GA-PSO) algorithm. In comparison to GA, the proposed algorithm efficiently distributes tasks among the resources, resulting in delays, waiting times, and energy consumption that are reduced by 8.73%, 22.65%, and 17.81%, respectively, and improved resource utilisation by 0.54%. When compared to the Round Robin algorithm, however, delays, energy consumption and waiting times were reduced by 3.90%, 1.68%, and 21.99% respectively, and improved resource utilisation was shown to be 12.51%. According to quantitative analysis, the proposed algorithm outperforms round-robin algorithms and GA and progresses toward ideal solutions more quickly.

In (14), Kumar et al. proposed the energy-saving Green-Demand Aware Fog Computing (GDAFC) solution. The suggested solution employs a prediction technique to determine the working fog nodes (nodes that respond to requests as they come in), standby fog nodes (nodes that step in when the working fog nodes' computational capacity is insufficient), and idle fog nodes in a fog computing infrastructure. Taking into account the delay requirements of the applications, it also assigns a suitable sleep interval for the fog nodes. Without impairing the performance of the delay requirement, this solution can save up to 65% of energy. The study has shown that it is not necessary to keep every fog node in an FCI active in order to deliver a requested service. The number of working nodes in an FCI is thus dynamically determined by the proposed technique based on the anticipated workload at a particular time. The system designates some of the fog nodes in an FCI as standby nodes to prevent task dropping or delay-requirement violations when the prediction accuracy is low.

In (15), Nguyen et al. introduce a method to reduce operating costs and execution times when solving task scheduling issues for Bag-of-Tasks applications in a cloud-fog environment. The TCaS proposed algorithm was evaluated on 11 datasets of varying sizes. In both a Fog environment and a Cloud-Fog system, TCaS outperformed three other methods, namely MPSO, BLA, and RR. The results demonstrate a 15.11% improvement over the Bee Life Algorithm (BLA) and an improvement of 11.04% over Modified Particle Swarm Optimization (MPSO) while accomplishing a harmony between completion time and operating expense. In terms of the trade-off between time and cost execution, RR increased by 44.17% over 11 sets of tasks, particularly by obtaining a significantly shorter scheduling length. The proposed method is cost-effective and has high-performance computing power.

In (16), Etemadi et al. proposed a technique that is mainly for IoT devices. IoT applications' time-varying workloads in the fog network should be managed using a learning-based resource provisioning approach, the proposal reads. Using the general three-tier architecture of fog networks as our model, we create an extended resource provisioning framework. The proposed method uses a hidden Markov model (HMM) as a decision-maker and a nonlinear autoregressive neural network (NAR) as a prediction method to decide how much fog resources to provision for IoT application workloads. Extension experiments are used to assess effectiveness using two real-world datasets. According to the iFogSim toolkit's results, resource energy consumption is improved while the delay, cost, and number of fog devices are reduced when compared to baseline mechanisms currently in use.

In (17), Duy La et al. using two case studies, suggest a strategy that uses device-

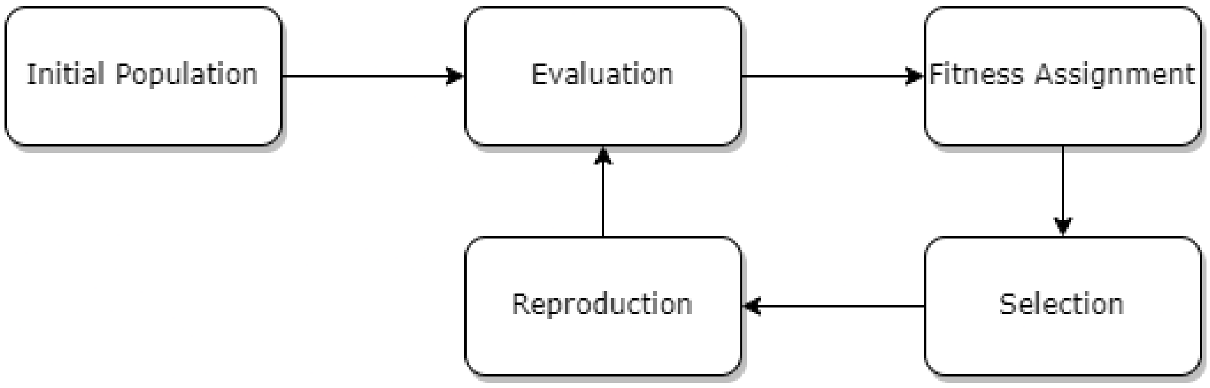


Figure 3: A simple Evolutionary Algorithm Architecture

driven and human-driven intelligence as essential enablers to lower energy consumption and latency in fog computing. The first is to perform adaptive low-latency Medium Access Control (MAC)-layer scheduling among sensor devices using machine learning to identify user behaviours and the other is to create an algorithm that will allow a smart EU device to choose its offloading decision in the company of several nearby fog nodes while minimising its own energy and latency goals. The first case study demonstrates how human-driven data analytics can enhance resource scheduling context awareness and network adaptability. The second case study shows that when task offloading is made possible by a group of fog nodes, the energy consumption and latency for the EU device are reduced.

3 Methodology

A subclass of evolutionary computation, the evolutionary algorithm (EA) is a general stochastic search algorithm. A metaheuristic optimisation algorithm based on the population concept was used. Metaheuristics are higher-level processes designed to locate, generate, or pick one or more lower-level processes or heuristics capable of performing partial searches. It can be used for a variety of optimisation problems with constrained computing power and incomplete or imperfect data. It offers a good enough solution in such circumstances(19).

The processes of biological evolution, such as reproduction, mutation, recombination, and selection, serve as inspiration for EAs. To maximise the quality function, a group of prospect solutions is generated at random. The problem domain is then applied with the quality function in the form of an abstract fitness function. On the basis of the fitness function, some better candidates are chosen for the next generation. By using the methods of recombination and/or mutation on them, this is accomplished. The binary operator can be used to represent recombination. This operator can be used to create one or more new candidates (children) by applying it to two or more chosen candidates known as parents. While a mutation only affects one candidate and produces a single new child. It generates a set of new candidates based on their fitness function after carrying out this recombination or mutation. This procedure is iterative. It can go on indefinitely as long as suitable candidates are found(19).

Below is how the evolutionary algorithms work:

1. Original problem space and problem-solving space are defined in the first step. It's referred to as representation. It is the environment in which evolution occurs. The goal of representation is to close the gap between real-world issues and the world of electronic games. Phenotypes are the items that make up the original problem space's potential solutions. The individuals within EA are referred to as genotypes according to their corresponding encoding(19).

2. Finding the evaluation function (Fitness Function) is the second step. This function serves as the foundation for the selection process and makes improvements possible(19).

3. Population once defined as representation holds a potential resolution. The unit of evolution is composed of multiple genotypes. The population is the total number of individuals in a given representation(19).

4. By choosing parents who are of high quality, the individuals are chosen. Because of this, the following generation has parents of high calibre(19).

5. Variation operators transform existing operators into new ones. Both mutation and recombination are types of variation operators. A unary operator serves as the mutation's representation. It creates the offspring of a genotype when it is applied to that genotype. Recombination, however, uses a binary operator. Recombination combines the information from two or more parent genotypes to produce one or more offspring genotypes(19).

6. Individuals are distinguished based on their merit through the survivor selection process. It resembles the process of choosing parents quite a bit. However, it occurs at various points in the evolution(19).

7. Individuals are chosen at random form the first population. Typically, an initial population is formed using a particular heuristic with a higher fitness. Reaching the ideal fitness level, which the issue is aware of, can cause the termination condition to occur. EA, however stochastic, does not ensure an ideal outcome. As a result, the algorithm never reaches the ideal fitness value. Therefore, it requires a condition that unquestionably stops the algorithm. The maximum CPU time, the total number of evaluations, the allotted time, etc. are some of the criteria used for the same(19).

Evolutionary Programming (EP), Evolution Strategies (ES), Genetic Programming (GP), and Genetic Algorithm(GA) are subfields of the evolutionary algorithm. Memetic algorithms (MA) and distributed EA are two additional extensions that are recommended to improve the overall performance of EA methods.

Local search heuristics are those that adopt a participant. The memetic algorithm (MA) combines the EA and local inquiry. Hill Climbing, Ant systems, Particle swarms, Simulated annealing, Differential Evolution, and Tabu Search are examples of search heuristics. Due to the possibility of a very slow evaluation of the fitness functions or excessively large population size, distributed EAs are used to distribute the entire workload across multiple computers and carry out all computations in parallel. The Master-slave model, Independent run model, island model, unified hypergraph model, hybrid model, and cellular EAs are examples of distributed EAs models(19).

Swarm intelligence, on the other hand, is also known as artificial groups of simple agents, and is a term that describes the joint behaviour of self-organized and decentralised techniques. Social insects' collective nest-building, cooperative transportation, group foraging, and collective sorting and clustering are all examples of swarm intelligence in action(21).

Metaheuristics called swarm optimisation algorithms were developed as a result of the swarm intelligence phenomenon. They are employed to address a variety of optimisation issues because of their ease of use, adaptability, and scalability. Particle swarm optimisation (PSO), ant colony optimisation (ACO), artificial bee colony (ABC), and grey wolf optimisation are a few of the well-known swarm optimisation algorithms (GWO). These algorithms search for the best solution in a given problem space by imitating the behaviour of natural swarms, such as birds, fish, bees, and wolves(21).

An optimisation algorithm works iteratively from a starting hypothesis. It might eventually unite towards a sturdy solution, ideally the best solution to the relevant issue, after a predetermined (sufficiently large) number of iterations. The solutions serve as states in this self-organizing system, and the converged solutions serve as attractors. A set of guidelines or a set of mathematical equations can guide the evolution of such an iterative, self-organizing system. As an outcome, an intricate system like this can interconnect and self-organize into specific converged forms, exhibiting some budding self-organizational traits. In this sense, designing an optimisation algorithm that is both effective and efficient is equivalent to figuring out how to mimic the evolution of a self-organizing system(20).

To determine which evolutionary and swarm-based algorithms work best in this situation, we will use a variety of them.

To simulate a cloud-fog environment, PureEdgeSim will be used. It is a simulation framework that allows the assessment of resource managing techniques as well as the performance of Cloud, Edge, and Mist Computing environments(22).

So as mentioned above we will be finding an efficient task-scheduling algorithm from several evolutionary and swarm-based algorithms.

4 Implementation

In this section, we are going to go through the practical implementation part of our research where we will be focusing on the various meta-heuristic algorithms for efficient task scheduling in a fog computing environment. First, we will be discussing the different algorithms we used in this research. Then we will move on to the simulator we use to imitate the Cloud fog environment and then we will discuss the different hardware specifications of the device used to run the simulator and algorithms.

4.1 Genetic Algorithm

The Genetic Algorithm (GA), developed by John Holland in the year of 1975, is a search optimisation technique based on way of natural selection. The primary idea behind this algorithm is to replicate the concept of the law of the jungle. A new population is generated in GA with the use of certain genetic operators such as crossover, reproduction, and mutation. A population can be represented as a string set (referred to as chromosomes). In each generation, a new chromosome (a member of the population) is formed utilising information from the preceding population's fittest chromosomes. GA first spawns a population of viable solutions and connects them in such a way that their search is directed toward more favourable portions of the search space. Each one of these viable solutions is concealed as a chromosome, sometimes known as a genotype, and each one of these chromosomes will be assigned a fitness function. The fitness function of a chromosome influences its ability to survive and reproduce progeny. The higher the fitness value, the more promising the solution is for maximising problems, and the lower the fitness value, the more useful the solution is for minimization problems. The five major components of a basic GA are a random number generator, a fitness evaluation unit, a reproduction process, a crossover process, and a mutation operation. Reproduction chooses the population's fittest candidates, whereas crossover is the process of merging the best chromosomes and passing the greatest genes to the next generation, and mutation changes part of the genes in a chromosome(21).

There are three variants of genetic algorithms implemented here, they are BaseGA, EliteSingleGA and EliteMultiGA.

4.2 Differential Evolution

The Differential Evolution (DE) algorithm is similar to GA and is a population-based method in that it uses the same operators, The key distinction between DE and GA is that DE depends on mutation operations while GA relies on crossover operations to build superior solutions. Storn and Price developed this algorithm in 1997. DE uses three attributes to generate a new population iteratively: Target Vector, Mutation Vector, and Trail Vector. The target vector is the vector that holds the answer to the search space; the target vector's mutation is the mutant vector; and the trailing vector is the outcome of the crossover operation between the target vector and the mutant vector. As previously noted, the primary steps of the DE algorithm are similar to GA with just minor variations. The population initialization process is where DE begins, followed by an evaluation to select the population's most fitting individuals. The weighted difference of the two population vectors is then added to the third vector to generate new parameter vectors. This process is known as mutation. The vector is blended within the crossover, and the algorithm performs a final selection step(21).

We will be adopting the Self-Adaptive Differential Evolution(SADE) algorithm, which is a variant of DE here.

Throughout the optimisation process, a self-adaptive differential evolution algorithm (SaDE) can automatically alter its mutation and crossover parameters and tactics. It does not necessitate that the user pre-specify these values, which can be difficult and time-consuming to optimise for various problems. SaDE can improve its performance and efficiency by learning from its own experience and adapting to the features of the situation at hand. SaDE employs a self-adaptive process to determine the best mutation approach and parameter values for each generation. It keeps a pool of mutation techniques and assigns each one a chance based on its past performance. Qin and Suganthan suggested SaDE in 2005 as a solution to overcome the constraints of the standard differential evolution (DE) algorithm(18).

4.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimisation approach developed in 1995 by Kennedy and Eberhart. To lead the particles in their search for global optimal solutions, it employs a simple technique that resembles swarm behaviour in birds flocking and fish schooling. PSO has three basic behaviours: separation, alignment, and cohesion. Separation is the avoidance of congested regional flockmates, whereas alignment is the movement toward the direction of regional flockmates. The behaviour of moving towards the average position of local flockmates is referred to as cohesion. By scanning an entire high-dimensional problem space, PSO has proven to be an efficient optimisation approach. It's a stable stochastic optimisation technique established on swarm movement and intelligence. It uses the notion of social exchange to solve problems and doesn't employ the gradient of the issue being optimised, hence it does not need the optimisation issue to be differential, as traditional optimisation methods do. PSO can be used to determine the optimisation of irregular issues that are noisy and dynamic. PSO parameters include the number of particles, the agent's position in the solution space, velocity, and agent neighbourhood(21).

4.4 Artificial Bee Colony

Among the most current swarm intelligence algorithms is Artificial Bee Colony(ABC). Dervis Karaboga proposed it in 2005. This algorithm is inspired by natural honey bees' cognitive behaviour in identifying food sources, called nectar and exchanging knowledge regarding that food source with other bees in the hive. The method is said to be as easy and straightforward to execute as DE and PSO. The artificial agents are classified into three sorts: employed bees, observer bees, and scout bees. Monitoring and remembering the location of that source is done by employed bees. Because each hired bee is associated with only one food source, the count of employed bees and food sources will be the same. The employed bee in the hive provides information on the food supply to the spectator bee. Following that, nectar is collected from one of the chosen food sources. The scout bee is responsible for discovering new food sources and nectar sources. The ABC method's main process and the details of each stage are as follows: Initialization Phase, Employed Bees Phase, Onlooker Bees Phase, Scout Bees Phase, Memorization of the best fitness value and position, and Termination Checking Phase(21).

4.5 Bacterial Foraging Optimization

Passino's Bacteria Foraging Optimization Algorithm (BFOA) is a newbie to the family of nature-inspired optimisation algorithms. The bacterial foraging optimisation algorithm (BFOA) is a bio-inspired optimisation technique that simulates the foraging behaviour of bacteria, such as Escherichia coli in search of nutrition. Chemotaxis, swarming, reproduction, and elimination-dispersal are the four processes of BFOA. Chemotaxis refers to microorganisms moving towards or away from chemical gradients. Bacteria swarming is the creation of persistent patterns in nutrient-rich settings. Bacteria reproduce by splitting into two identical cells. The abrupt change in bacterium sites caused by environmental changes is known as elimination dispersal. By altering the settings and methods of these processes, BFOA may handle both continuous and discrete optimisation problems(23).

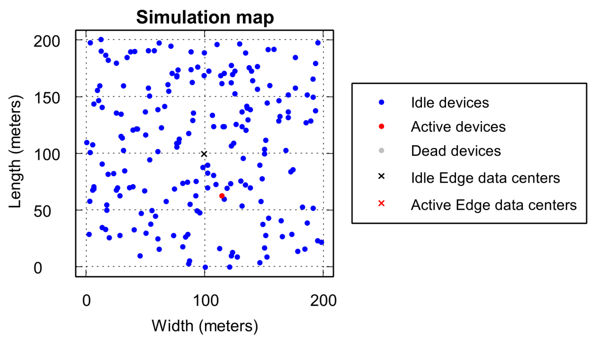


Figure 4: Simulation Map

4.6 Bees Algorithm

Another bio-inspired optimisation tool is the Bees algorithm (BA), which simulates the food-foraging behaviour of honey bees. BA has three types of bees: hired bees, bystanders, and scouts. Employed bees are affiliated with a specific food source (solution) and communicate with bystanders. Onlookers are bees who wait in the hive and choose a food source based on information provided by hired bees. Scouts are bees that randomly explore new food sources in the search space. BA can strike a balance between exploration and exploitation by employing different search algorithms for different bee groupings.

The probabilistic bees algorithm (PBA) is a variation of the BA that incorporates probabilistic models to improve the algorithm's performance and efficiency. PBA uses an estimated distribution algorithm (EDA) to learn from previous solutions and generate new ones based on a probability distribution. PBA additionally employs a self-adaptive mechanism to dynamically alter the algorithm's settings and tactics based on the features of the challenge(24).

4.7 Cat Swarm Optimization

The cat swarm optimisation algorithm (CSO) is a swarm-based optimisation technique inspired by feline behaviour. CSO operates in two modes: searching and tracing. Cats in seeking mode are passive and look about for probable prey. Cats are active and hunt after their prey when in tracing mode. CSO controls the searching mode using four factors: seeking memory pool (SMP), seeking range of the selected dimension (SRD), counts of dimension to change (CDC), and self-position considering (SPC). Each cat's memory size is determined by SMP. SRD calculates the mutation ratio for the dimensions chosen. CDC specifies how many dimensions will be altered. SPC determines whether a cat's current posture is a candidate point or not. Using various distance metrics, CSO can tackle both combinatorial and continuous optimisation problems(25).

A simulation framework that allows the assessment of resource managing techniques as well as the performance of Cloud, Edge, and Mist Computing environments. Every aspect of Edge Computing's modelling and simulation is covered. The modular architecture of PureEdgeSim deals with a distinct element of the simulation with each module. The Network Module, for instance, handles bandwidth allotment and data transport. The Location Management module addresses the geo-allotment of devices and their mobility. The Data Centers Manager module manages the heterogeneity of the device generation.

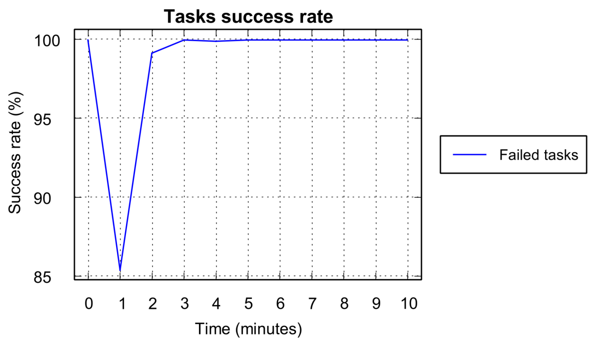


Figure 5: Task Success Rate

The last module is the Orchestrator, which evaluates the offloading of tasks. These modules also contain a default implementation and a collection of modifiable parameters to aid with investigation and prototyping. The simulation's outcomes show how effective PureEdgeSim is at recreating complex, varied, and dynamic settings. Also, they stress the benefits of putting Mist Computing into practice and the usefulness of the suggested approach, which outperformed the alternatives in every test situation(22).

With classical cloud computing, devices at the edge of the network unload their duties to the cloud for processing. For a variety of reasons, including the restricted computing power of particular devices and the requirement to increase battery life for devices with capacity-limited batteries, this task offloading may be necessary. Edge and Mist Computing both utilise the identical offloading process. By offloading duties, edge nodes can work together to improve system throughput. A related technique called Serendipity allows mobile devices to remotely use the resources of other devices to execute their apps, using much less local power and completing tasks 6.6 times faster. EdgeCloudSim is a mobile edge computing simulator built on CloudSim that fixes some of the issues with iFogSim. Since it automatically develops the needed number of edge devices, it is more scalable. The network model is more precise, and it does encourage mobility to some extent(22)

For experiments, we have used an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (2.59 GHz) and 16 GB of main memory. The meta-heuristic algorithms were implemented in Python using the mealpy and other libraries so that the application module could obtain the optimum scheduling policy. This generated policy from the algorithms is then passed to PureEdgeSim which simulates the fog system.

5 Evaluation

Figure 7 gives the results from the simulation using various algorithms.

5.1 Discussion of Results

In the experiment, the performance of various meta-heuristic algorithms namely the Base Genetic Algorithm(BaseGA), Elite Single Genetic Algorithm(EliteSingleGA), Elite Multi Genetic Algorithm(EliteMultiGA), Self-Adaptive Differential Evolution(SADE) Algorithm, Original Particle Swarm Optimisation(OriginalPSO) Algorithm, Original Bac-

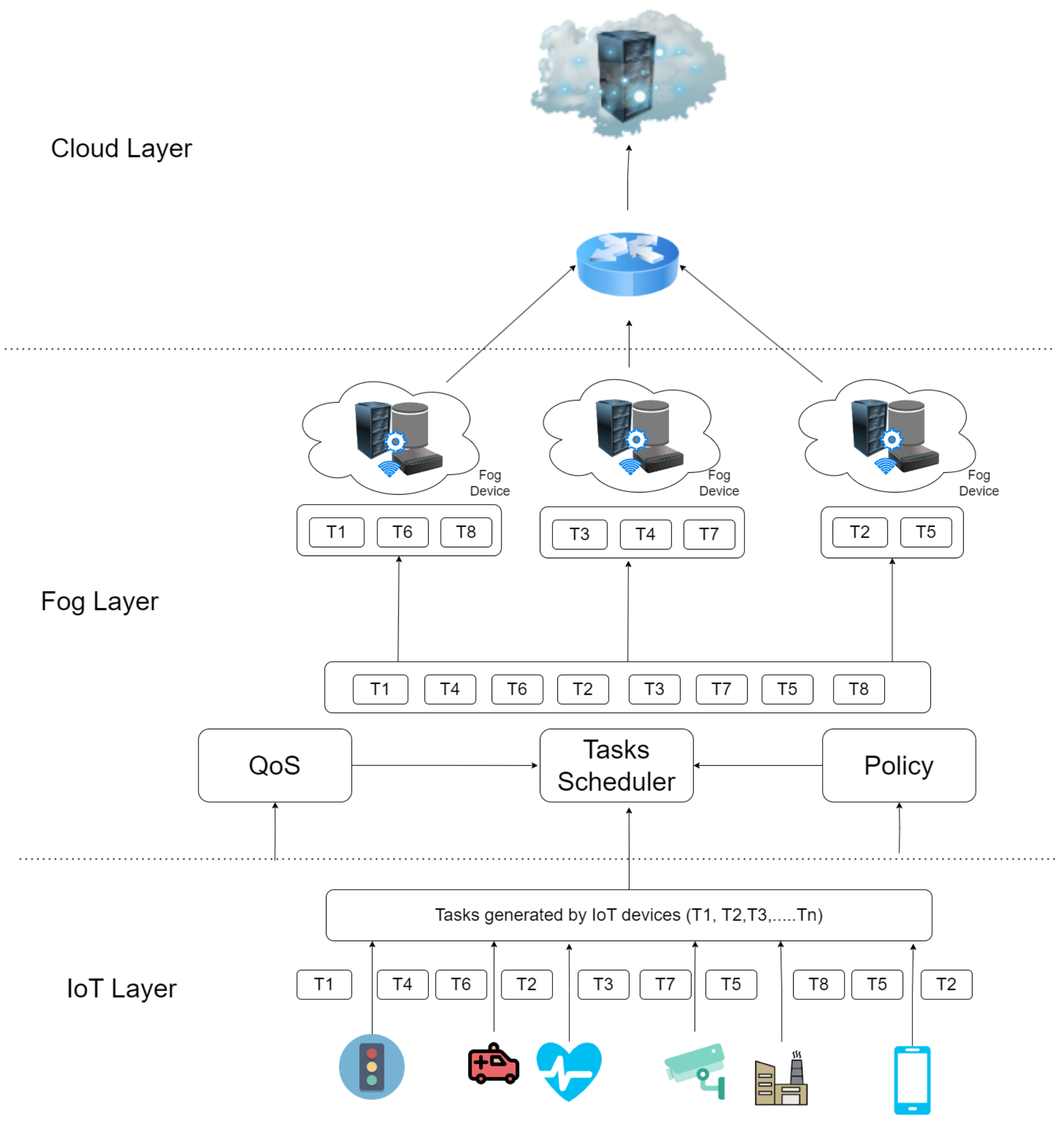
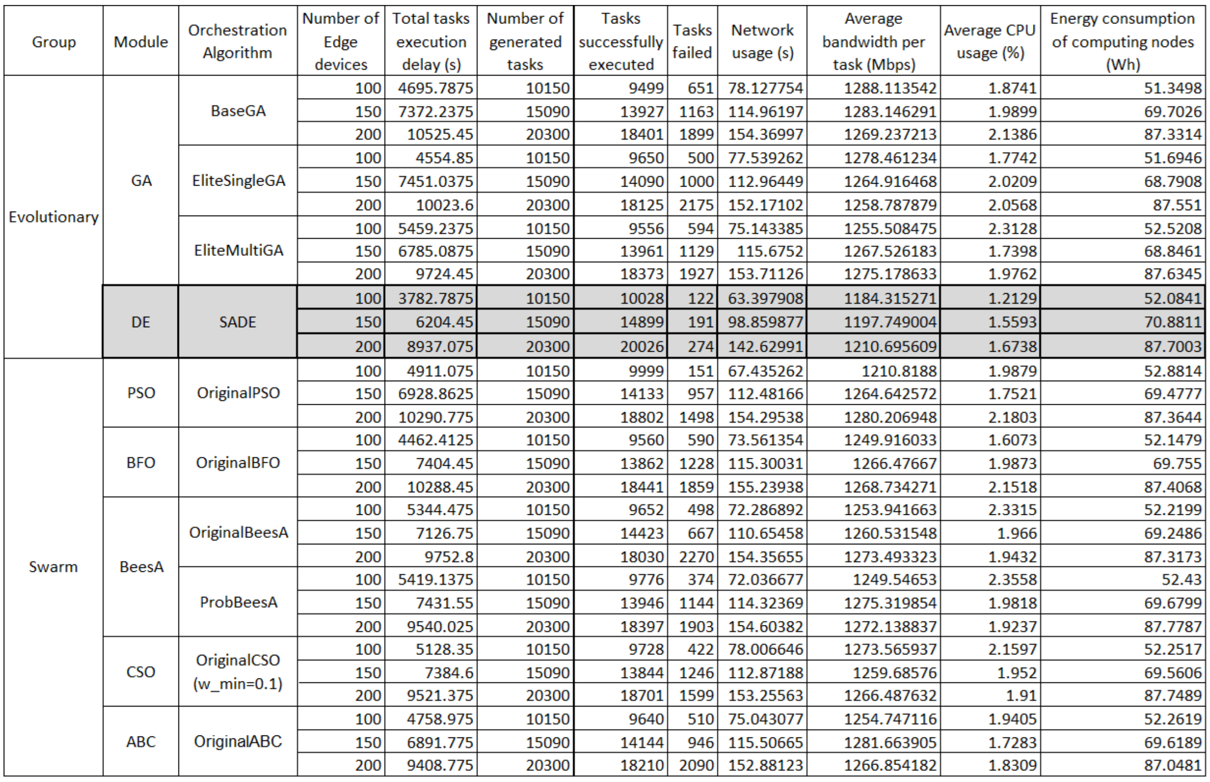


Figure 6: Architecture of the Task Scheduling in Cloud Fog environment

terial Foraging Optimization(OriginalBFO) Algorithm, Original Bees Algorithm(OriginalBeesA), Probabilistic Bees Algorithm(ProbBeesA), Original Cat Swarm Optimisation(OriginalCSO) Algorithm, and Original Artificial Bee Colony(OriginalABC) Algorithm are compared with each other to find the algorithm which performs the best. They are evaluated in terms of task execution delay, number of successful/failed tasks, network usage, average bandwidth per task, average CPU usage and energy consumption.

The graphs are created based on the results of the simulation in Figure 7. Here we have chosen the results when the number of edge devices is 200 to create the graphs.

Figure 8 shows the graph of the number of tasks successfully executed by several algorithms. Out of a total number of 20300 tasks, the number of successful tasks varies from 18030 for OriginalBeesA to 20026 for SADE. Along with that, comparing the successful



Group	Module	Orchestration Algorithm	Number of Edge devices	Total tasks execution delay (s)	Number of generated tasks	Tasks successfully executed	Tasks failed	Network usage (s)	Average bandwidth per task (Mbps)	Average CPU usage (%)	Energy consumption of computing nodes (Wh)
Evolutionary	GA	BaseGA	100	4695.7875	10150	9499	651	78.127754	1288.113542	1.8741	51.3498
			150	7372.2375	15090	13927	1163	114.96197	1283.146291	1.9899	69.7026
			200	10525.45	20300	18401	1899	154.36997	1269.237213	2.1386	87.3314
		EliteSingleGA	100	4554.85	10150	9650	500	77.539262	1278.461234	1.7742	51.6946
			150	7451.0375	15090	14090	1000	112.96449	1264.916468	2.0209	68.7908
			200	10023.6	20300	18125	2175	152.17102	1258.787879	2.0568	87.551
		EliteMultiGA	100	5459.2375	10150	9556	594	75.143385	1255.508475	2.3128	52.5208
			150	6785.0875	15090	13961	1129	115.6752	1267.526183	1.7398	68.8461
			200	9724.45	20300	18373	1927	153.71126	1275.178633	1.9762	87.6345
	DE	SADE	100	3782.7875	10150	10028	122	63.397908	1184.315271	1.2129	52.0841
			150	6204.45	15090	14899	191	98.859877	1197.749004	1.5593	70.8811
			200	8937.075	20300	20026	274	142.62991	1210.695609	1.6738	87.7003
Swarm	PSO	OriginalPSO	100	4911.075	10150	9999	151	67.435262	1210.8188	1.9879	52.8814
			150	6928.8625	15090	14133	957	112.48166	1264.642572	1.7521	69.4777
			200	10290.775	20300	18802	1498	154.29538	1280.206948	2.1803	87.3644
	BFO	OriginalBFO	100	4462.4125	10150	9560	590	73.561354	1249.916033	1.6073	52.1479
			150	7404.45	15090	13862	1228	115.30031	1266.47667	1.9873	69.755
			200	10288.45	20300	18441	1859	155.23938	1268.734271	2.1518	87.4068
	BeesA	OriginalBeesA	100	5344.475	10150	9652	498	72.286892	1253.941663	2.3315	52.2199
			150	7126.75	15090	14423	667	110.65458	1260.531548	1.966	69.2486
			200	9752.8	20300	18030	2270	154.35655	1273.493323	1.9432	87.3173
		ProbBeesA	100	5419.1375	10150	9776	374	72.036677	1249.54653	2.3558	52.43
			150	7431.55	15090	13946	1144	114.32369	1275.319854	1.9818	69.6799
			200	9540.025	20300	18397	1903	154.60382	1272.138837	1.9237	87.7787
	CSO	OriginalCSO (w_min=0.1)	100	5128.35	10150	9728	422	78.006646	1273.565937	2.1597	52.2517
			150	7384.6	15090	13844	1246	112.87188	1259.68576	1.952	69.5606
			200	9521.375	20300	18701	1599	153.25563	1266.487632	1.91	87.7489
	ABC	OriginalABC	100	4758.975	10150	9640	510	75.043077	1254.747116	1.9405	52.2619
			150	6891.775	15090	14144	946	115.50665	1281.663905	1.7283	69.6189
			200	9408.775	20300	18210	2090	152.88123	1266.854182	1.8309	87.0481

Figure 7: Table containing results of the simulation

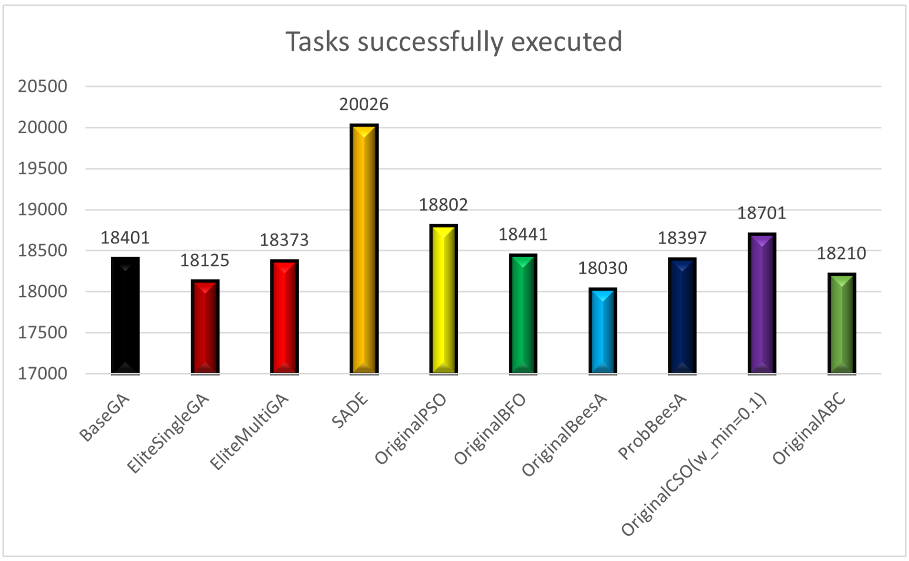


Figure 8: Graph comparing number of successful tasks

task count of other algorithms we have BaseGA with 18401 tasks, EliteSingleGA with 18125 tasks, EliteMultiGA with 18373 tasks, OriginalPSO with 18802 tasks, OriginalBFO with 18441 tasks, ProbBeesA with 18397 tasks, OriginalCSO with 18701 tasks and OriginalABC with 18210 tasks. Comparing the values of all the algorithms, the SADE algorithm performs much better than the other algorithm with a difference of 1224 than the second best performing one.

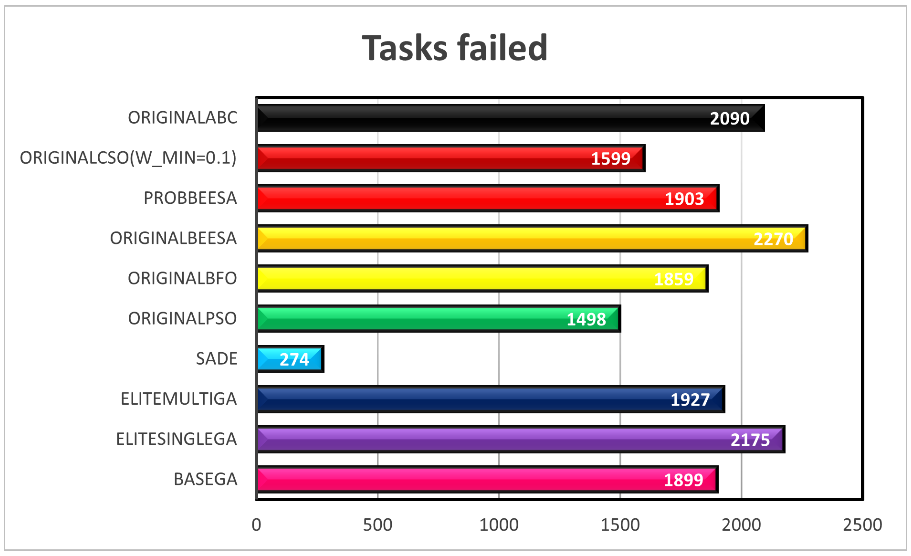


Figure 9: Graph comparing number of failed tasks

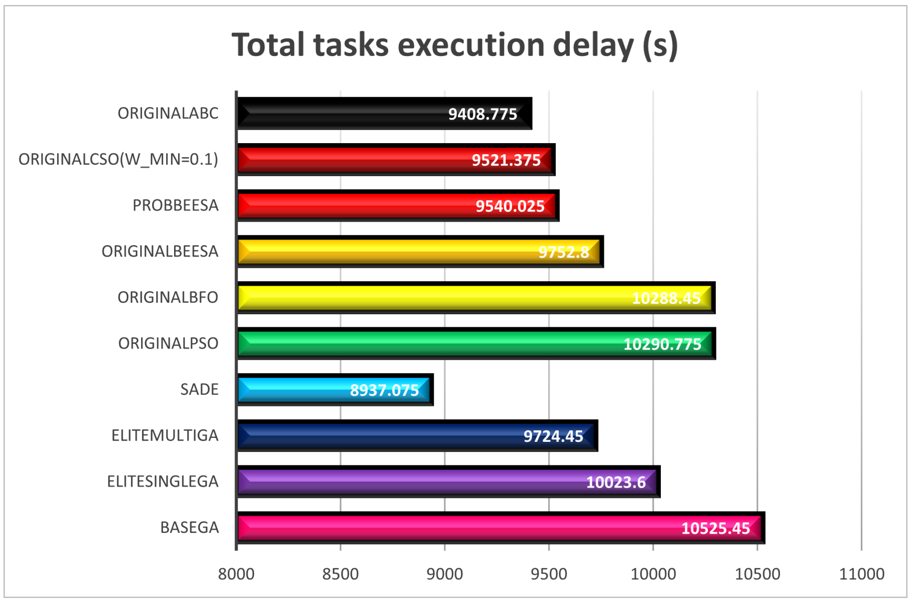


Figure 10: Graph comparing total tasks execution delay

Similarly, Figure 9 shows the graph of the number of tasks failed by several algorithms. Out of a total number of 20300 tasks, the number of failed tasks varies from 2270 for OriginalBeesA to 274 for SADE. Along with that, comparing the successful task count of other algorithms we have BaseGA with 1899 tasks, EliteSingleGA with 2175 tasks, EliteMultiGA with 1927 tasks, OriginalPSO with 1498 tasks, OriginalBFO with 1859 tasks, ProbBeesA with 1903 tasks, OriginalCSO with 1599 tasks and OriginalABC with 2090 tasks. Comparing the values of all the algorithms, the SADE algorithm performs much better than the other algorithm with a difference of 1224 than the second best performing one.

Similarly, Figure 10 shows the graph plotting execution delay for all the tasks of each of the algorithms. SADE is the best-performing algorithm with the least delay of

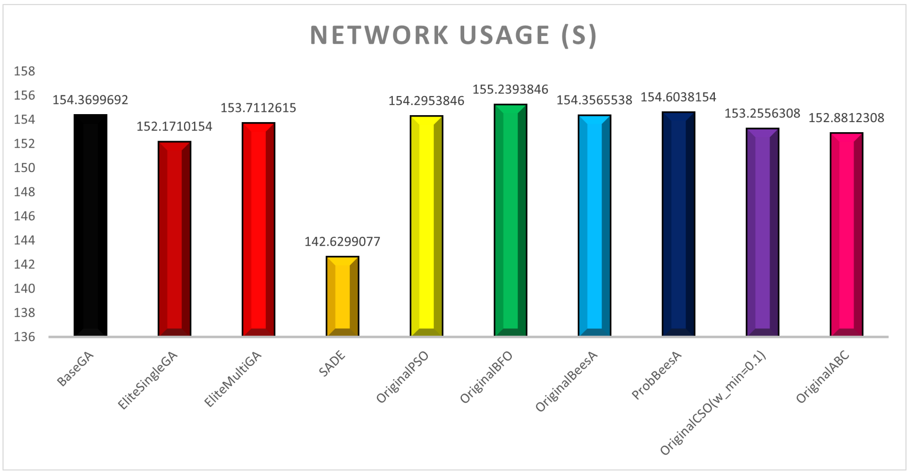


Figure 11: Graph comparing network usage

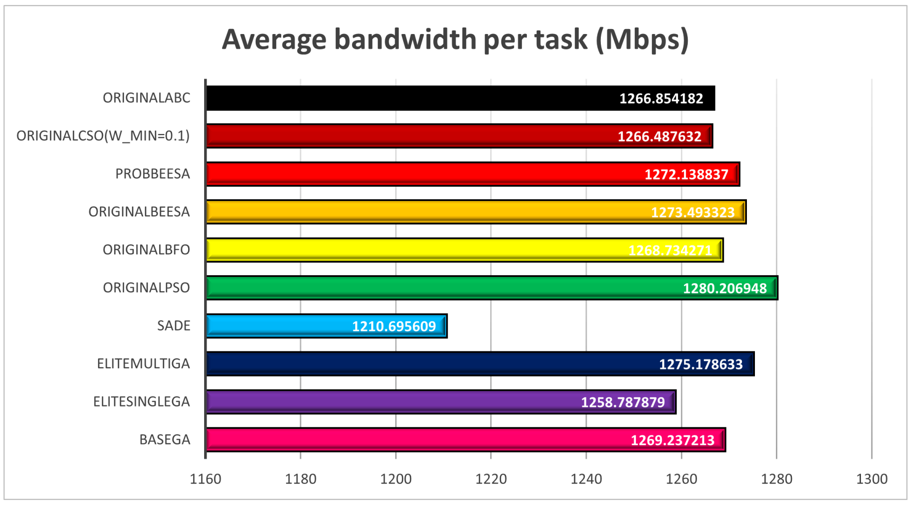


Figure 12: Graph comparing average bandwidth per task

8937.075 seconds while BaseGA performs worst with a total delay of 10525.45 seconds. Comparing the delay for the rest of them, we have EliteSingleGA with 10023.6 seconds, EliteMultiGA with 9724.45 seconds, OriginalPSO with 10290.775 seconds, OriginalBFO with 10288.45 seconds, OriginalBeesA with 9752.8 seconds, ProbBeesA with 9540.025 seconds, OriginalCSO with 9521.375 seconds and OriginalABC with 9408.775 seconds. The total delay for the SADE algorithm is much lower than all the other ones thus performing much better than the rest.

Similarly, Figure 11 shows the graph plotting network usage for all the tasks of each of the algorithms. SADE is the best-performing algorithm with the least network usage of 142.629 S while OriginalBFO performs worst with network usage of 155.239 S. Comparing the network usage for the rest of them, we have BaseGA with 154.359 EliteSingleGA with 152.171 S, EliteMultiGA with 153.711 S, OriginalPSO with 154.295 S, OriginalBeesA with 154.356 S, ProbBeesA with 154.603 S, OriginalCSO with 153.255 S and OriginalABC with 152.881 S. The network usage for the SADE algorithm is much lower than all the other ones thus performing much better than the rest.

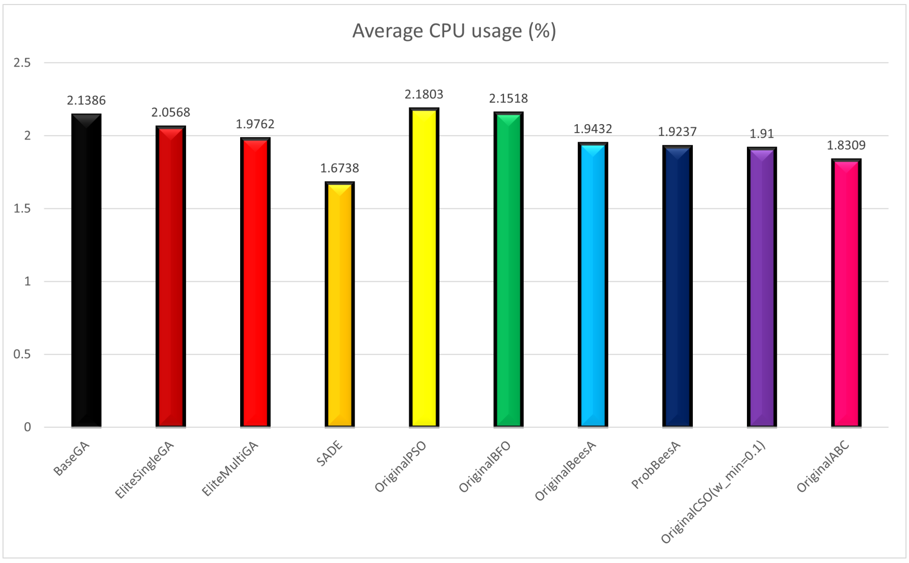


Figure 13: Graph comparing average CPU usage

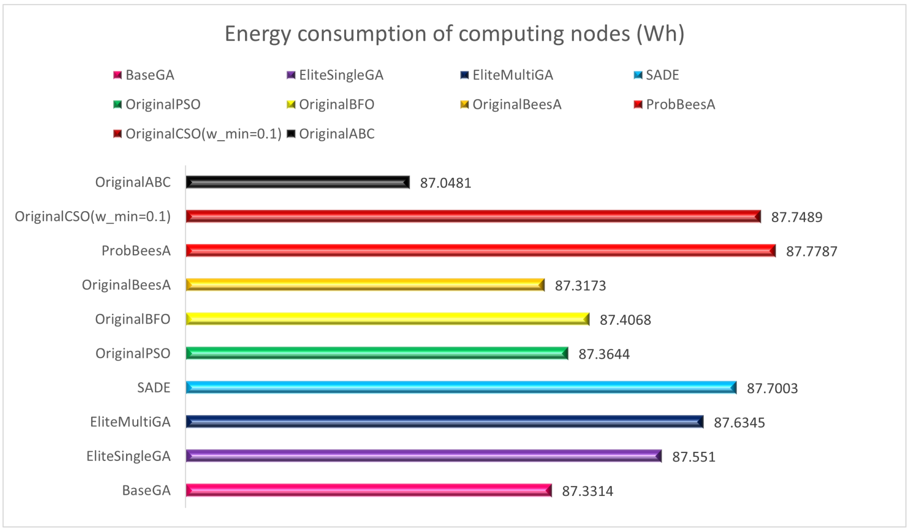


Figure 14: Graph comparing energy consumption

Similarly, Figure 12 shows the graph plotting the average bandwidth per task of each of the algorithms. SADE is the best-performing algorithm with the least average bandwidth of 1210.695 Mbps while OriginalPSO performs worst with an average bandwidth of 1280.206 Mbps. Comparing the average bandwidth for the rest of them, we have BaseGA with 1269.237 Mbps, EliteSingleGA with 1258.787 Mbps, EliteMultiGA with 1275.178 Mbps, OriginalBFO with 1268.734 Mbps, OriginalBeesA with 1273.493 Mbps, ProbBeesA with 1272.138 Mbps, OriginalCSO with 1266.487 Mbps and OriginalABC with 1266.854 Mbps. The average bandwidth per task for the SADE algorithm is much lower than all the other ones thus performing much better than the rest.

Similarly, Figure 13 shows the graph plotting average CPU usage for all the tasks of each of the algorithms. SADE is the best-performing algorithm with the least CPU usage of 1.6738% while OriginalPSO performs worst with an average CPU usage of

2.1803%. Comparing the CPU usage for the rest of them, we have BaseGA with 2.1386%, EliteSingleGA with 2.0568%, EliteMultiGA with 1.9762%, OriginalBFO with 2.1518%, OriginalBeesA with 1.9432%, ProbBeesA with 1.9237%, OriginalCSO with 1.91% and OriginalABC with 1.8309%. The average CPU usage for the SADE algorithm is much lower than all the other ones thus performing much better than the rest.

Similarly, Figure 14 shows the graph of energy consumption of computing nodes by several algorithms. Out of all the algorithms used, comparing the energy consumption of each algorithm we have BaseGA with 87.331, EliteSingleGA with 87.551, EliteMultiGA with 87.634, SADE with 37.7, OriginalPSO with 87.364, OriginalBFO with 87.406, OriginalBeesA with 87.317, ProbBeesA with 87.778, OriginalCSO with 87.748 and OriginalABC with 87.048. Comparing the values of all the algorithms, the OriginalABC algorithm performs much better than the other algorithm but there is only a negligible difference between the energy consumption of computing nodes of all the algorithms.

5.2 Observations

From the results, we saw that some algorithms specifically SADE algorithms perform better than the rest of the algorithms. One thing that has been observed during the simulations is that when using some of those meta-heuristic algorithms when the number of tasks successfully executed increases, tasks allocated to the cloud increase and tasks allocated to edge and mist decrease. At the same time, for some algorithms when there are fewer tasks allocated to the cloud and more to edge and mist devices, the number of tasks failed increases.

While on execution delay SADE has a decrease of 15.09% compared to BaseGA, OriginalBeesA has an increase of 11.07% on the number of successful tasks and a decrease of 87.93% when it comes to the number of failed tasks. While compared with OriginalPSO, SADE has a decrease of 5.43% in average bandwidth per task and a decrease of 23.23% in average CPU usage. Also on network usage, SADE has a decrease of 8.12% compared to OriginalBFO and on energy consumption, SADE only has an increase of 0.27% compared to the average of all other algorithms.

6 Conclusion and Future Work

The Internet of Things(IoT) is a technology that is rapidly developing day by day. The scale of growth is so huge, that it is being adopted into almost everywhere that it is now present in every part of our life. The more it grows, the more the resources to adopt it increase, whether its storage or computational and processing power. Cloud was a good solution at one point, but now that IoT is being integrated into real-time and latency-sensitive applications cloud computing is not enough anymore and that's how we adopted cloud fog environments. However, due to the heterogeneous nature of fog devices, it is important to have a good task-scheduling algorithm in fog computing. Out of the multiple meta-heuristic algorithms that we experimented with, differential evolution or specifically Self-Adaptive Differential Evolution(SADE) had the best performance. The results show that SADE based solution improves the performance by 5-88% when it comes to execution delay, successful/failed tasks, network usage, average bandwidth, and CPU usage. And also although it doesn't have the best performance when it comes to energy consumption, it showed that values were quite good.

In the future, it will be a good decision to find an algorithm that can decrease energy consumption as it is key in today's world. Another suggestion to keep in mind for the future is to have an algorithm that would decrease the number of tasks assigned to the cloud without compromising the tasks' success.

References

[1] McKinsey (2015). By 2025, Internet of things applications could have $11 trillion impact. [online] McKinsey & Company. Available at: https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact.

[2] Accenture (2015). Industrial Internet of Things Will Boost Economic Growth but Greater Government and Business Action Needed to. [online] Accenture.com. Available at: https://newsroom.accenture.com/news/industrial-internet-of-things-will-boost-economic-growth-but-greater-government-and-business-action-needed-to-fulfill-its-potential-finds-accenture.htm.

[3] Atlam, H., Walters, R. and Wills, G. (2018). Fog Computing and the Internet of Things: A Review. Big Data and Cognitive Computing, [online] 2(2), p.10. doi:https://doi.org/10.3390/bdcc2020010.

[4] Dastjerdi, A.V. and Buyya, R. (2016). Fog Computing: Helping the Internet of Things Realize Its Potential. Computer, 49(8), pp.112–116. doi:https://doi.org/10.1109/mc.2016.245.

[5] Nabi, S. and Ahmed, M. (2021). OG-RADL: overall performance-based resource-aware dynamic load-balancer for deadline constrained Cloud tasks. The Journal of Supercomputing, 77(7), pp.7476–7508. doi:https://doi.org/10.1007/s11227-020-03544-z.

[6] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S. (2012). Fog computing and its role in the internet of things. Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12. doi:https://doi.org/10.1145/2342509.2342513.

[7] Jawad Usman Arshed, Ahmed, M., Muhammad, T., Afzal, M., Arif, M. and Banchigize Mekcha Bazezew (2022). GA-IRACE: Genetic Algorithm-Based Improved Resource Aware Cost-Efficient Scheduler for Cloud Fog Computing Environment. 2022, pp.1–19. doi:https://doi.org/10.1155/2022/6355192.

[8] Yang, X. and Rahmani, N. (2020). Task scheduling mechanisms in fog computing: review, trends, and perspectives. Kybernetes, ahead-of-print(ahead-of-print). doi:https://doi.org/10.1108/k-10-2019-0666.

[9] Natesha, B.V. and Guddeti, R.M.R. (2021). Adopting elitism-based Genetic Algorithm for minimizing multi-objective problems of IoT service placement in fog computing environment. Journal of Network and Computer Applications, p.102972. doi:https://doi.org/10.1016/j.jnca.2020.102972.

[10] Farooq Hoseiny, Azizi, S., Shojafar, M., Fardin Ahmadiazar and Rahim Tafazolli (2021). PGA: A Priority-aware Genetic Algorithm for Task Scheduling in Heterogeneous Fog-Cloud Computing. Conference on Computer Communications Workshops. doi:https://doi.org/10.1109/infocomwkshps51825.2021.9484436.

[11] Abdel-Basset, M., Mohamed, R., Chakrabortty, R.K. and Ryan, M.J. (2021). IEGA: An improved elitism-based genetic algorithm for task scheduling problem in fog computing. International Journal of Intelligent Systems. doi:https://doi.org/10.1002/int.22470.

[12] Ali, I.M., Sallam, K.M., Moustafa, N., Chakraborty, R., Ryan, M.J. and Choo, K.-K.R. (2020). An Automated Task Scheduling Model using Non-Dominated Sorting Genetic Algorithm II for Fog-Cloud Systems. IEEE Transactions on Cloud Computing, pp.1–1. doi:https://doi.org/10.1109/tcc.2020.3032386.

[13] Singhrova, A. (2020). PRIORITIZED GA-PSO ALGORITHM FOR EFFICIENT RESOURCE ALLOCATION IN FOG COMPUTING. Indian Journal of Computer Science and Engineering, 11(6), pp.907–916. doi:https://doi.org/10.21817/indjcse/2020/v11i6/201106205.

[14] Pg. Ali Kumar, Dk.S.N.K., Newaz, S.H.S., Rahman, F.H., Lee, G.M., Karmakar, G. and Au, T.-W. (2022). Green Demand Aware Fog Computing: A Prediction-Based Dynamic Resource Provisioning Approach. Electronics, 11(4), p.608. doi:https://doi.org/10.3390/electronics11040608.

[15] Nguyen, B.M., Thi Thanh Binh, H., The Anh, T. and Bao Son, D. (2019). Evolutionary Algorithms to Optimize Task Scheduling Problem for the IoT Based Bag-of-Tasks Application in Cloud–Fog Computing Environment. Applied Sciences, 9(9), p.1730. doi:https://doi.org/10.3390/app9091730.

[16] Etemadi, M., Ghobaei-Arani, M. and Shahidinejad, A. (2020). A learning-based resource provisioning approach in the fog computing environment. Journal of Experimental & Theoretical Artificial Intelligence, pp.1–24. doi:https://doi.org/10.1080/0952813x.2020.1818294.

[17] La, Q.D., Ngo, M.V., Dinh, T.Q., Quek, T.Q.S. and Shin, H. (2019). Enabling intelligence in fog computing to achieve energy and latency reduction. Digital Communications and Networks, [online] 5(1), pp.3–9. doi:https://doi.org/10.1016/j.dcan.2018.10.008.

[18] Mahamed, Salman, A.A. and Engelbrecht, A.P. (2005). Self-adaptive Differential Evolution. pp.192–199. doi:https://doi.org/10.1007/11596448_28.

[19] Vikhar, P.A. (2016). Evolutionary algorithms: A critical review and its future prospects. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICGTSPICC.2016.7955308.

[20] Yang, X.-S. (2013). Swarm intelligence based algorithms: a critical analysis. Evolutionary Intelligence, 7(1), pp.17–28. doi:https://doi.org/10.1007/s12065-013-0102-2.

[21] Ab Wahab, M.N., Nefti-Meziani, S. and Atyabi, A. (2015). A Comprehensive Review of Swarm Optimization Algorithms. PLOS ONE, 10(5), p.e0122827. doi:https://doi.org/10.1371/journal.pone.0122827.

[22] Mechalikh, C., Taktak, H. and Moussa, F. (2019). PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments. 2019 International Conference on High Performance Computing & Simulation (HPCS), [online] pp.700–707. doi:https://doi.org/10.1109/HPCS48598.2019.9188059.

[23] Das, S., Biswas, A., Dasgupta, S. and Abraham, A. (2009). Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications. Foundations of Computational Intelligence Volume 3, pp.23–55. doi:https://doi.org/10.1007/978-3-642-01085-9_2.

[24] Bahari, M.S., Nur Athirah Azmi, Zahayu Md Yusof and Duc Truong Pham (2021). Bees Algorithm with Integration of Probabilistic Models for Global Optimization. Springer eBooks, pp.269–277. doi:https://doi.org/10.1007/978-981-16-0866-7_22.

[25] Yuce, B., Packianather, M., Mastrocinque, E., Pham, D. and Lambiase, A. (2013). Honey Bees Inspired Optimization Method: The Bees Algorithm. Insects, 4(4), pp.646–662. doi:https://doi.org/10.3390/insects4040646.