INCUBATION OF METAHEURISTIC SEARCH ALGORITHMS INTO NOVEL APPLICATION FIELDS

WILLA ARIELA SYAFRUDDIN

Preface

Several optimization algorithms have been developed to handle various optimization issues in many fields, capturing the attention of many researchers. Algorithm optimizations are commonly inspired by nature or involve the modification of existing algorithms. So far, the new algorithms are set up and focusing on achieving the desired optimization goal. While this can be useful and efficient in the short term, in the long run, this is not enough as it needs to repeat for any new problem that occurs and maybe in specific difficulties, therefore one algorithm cannot be used for all real-world problems.

This dissertation provides three approaches for implementing metaheuristic search (MHS) algorithms in fields that do not directly solve optimization issues. The first approach is to study parametric studies on MHS algorithms that attempt to understand how parameters work in MHS algorithms. In this first direction, we choose the Jaya algorithm, a relatively recent MHS algorithm defined as a method that does not require algorithm-specific control parameters. In this work, we incorporate weights as an extra parameter to test if Jaya's approach is actually "parameter-free." This algorithm's performance is evaluated by implementing 12 unconstrained benchmark functions. The results will demonstrate the direct impact of parameter adjustments on algorithm performance.

The second approach is to embed the MHS algorithm on the Blockchain Proof of Work (PoW) to deal with the issue of excessive energy consumption, particularly in using bitcoin. This study uses an iterative optimization algorithm to solve the Traveling Salesperson Problem (TSP) as a model problem, which has the same concept as PoW and requires extending the Blockchain with additional blocks. The basic idea behind this research is to increase the tour cost for the best tour found for n blocks, extended by adding one more city as

a requirement to include a new block in the Blockchain. The results reveal that the proposed concept can improve the way the current system solves complicated cryptographic problems

Furthermore, MHS are implemented in the third direction approach to solving agricultural problems, especially the cocoa flowers pollination. We chose the problem in pollination in cacao flowers since they are distinctive and different from other flowers due to their small size and lack of odor, allowing just a few pollinators to successfully pollinate them, most notably a tiny midge called *Forcipomyia Inornatipennis* (FP). This concept was then adapted and implemented into an Idle-Metaheuristic for simulating the pollination of cocoa flowers. We analyze how MHS algorithms derived from three well-known methods perform when used to flower pollination problems. Swarm Intelligence Algorithms, Individual Random Search, and Multi-Agent Systems Search are the three methodologies studied here. The results shows that the Multi-Agent System search performs better than other methods.

The findings of the three approaches reveal that adopting an MHS algorithms can solve the problem in this study by indirectly solving the optimization problem using the same problem model concept. Furthermore, the researchers concluded that parameter settings in the MHS algorithms are not so difficult to use, and each parameter can be adjusted to solve the real-world issue. This study is expected to encourage other researchers to improve and develop the performance of MHS algorithms used to deal with multiple real-world problems.

Acknowledgment

I am greatly indebted and would like to acknowledge the following individuals:

- I want to thank Prof. Mario Köppen for the guidance, supervision, and support during the doctoral course at Kyushu Institute of Technology, Japan. I also would like to thank Prof. Kaori Yoshida for her guidance and advice during my study in Japan.

- I am also indebted to the members of the graduation committee, Prof. Masato Tsuru, Prof. Kei Ohnishi, Prof. Xiaoqing Wen that spent much time reading my manuscript and giving constructive comments, Prof. Miroslav Hudec, that attended my defense presentation even though we had a very different time difference and gave me much helpful advice to improve my dissertation.

- I am thankful for the financial supports from Kyutech Foundation Scholarship and the KIT tuition fee exemption. Also, I thank fellow researchers and staff: Brahim Benaissa, Sajjad Dadkhah, Kobayashi-san, Apiradee-san, Ogata-san, Ito-san, Miwa-san, and many others with whom I was involved in many discussions and provided assistance in many ways during my stay in Japan.

- I cannot thank enough all fellow lab members, fellow Indonesian students in Kyutech, fellow International students in Kyutech, Fatini, Hafizah, and many others who always supported and provided me with a good balance with out-of-work life.

- To whom undoubtedly I am indebted most, and I would like to dedicate this work to my beloved parents, my sisters, my brother-in-law, my nieces and nephews for the never-ending support and the extra bit of motivation to finish the doctoral course.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Optimization being a method that is widely used even in our daily lives, we earn many benefits from the use of mathematical optimization techniques. Optimization is commonly used in diverse fields, from engineering design to computer science, such as GPS systems, scheduling, and economics, like package delivery businesses and financial organizations. We are often trying to enhance or minimize anything. However, knowing that everything is optimized makes troubleshooting more complex. Many seemingly simple problems are, in reality, quite tough to solve. In reality, we are always searching for the best solution to every situation we encounter, even though we may not always find such solutions.

An optimization algorithm is a group of mathematical algorithm or methods that try to discover the minimum value of a mathematical function for use in machine learning or other engineering fields. Optimization algorithms are primarily used to solve optimization problems. Many researchers propose various types of new optimization algorithms to solve various types of optimization problems. The majority of these algorithms are inspired by nature, such as the behavior of animals looking for food, and some known algorithms include the one proposed by the authors of Karaboga and Bahriye [1] describing the Artificial Bee Colony (ABC) Algorithm, which is inspired by how the bees working together to collect the food. Authors Yan and Deb [2] also propose in their research that Cuckoo Search via Levy Flight is based on the breeding behavior of some cuckoo species combined with

the Levy flight behavior of some birds and fruit flies.

Optimization algorithms are essential because optimization problems can consult in various scientific fields such as economics, engineering, and medicine. There are still many researchers around the world working to solve problems in this field. Although the classical optimization algorithm approach can handle some real-world issues, in some circumstances, the classical method is inefficient in handling real-world problems because it cannot discover global optima, necessitating some modification as a solution. In contrast, metaheuristic search (MHS) algorithms are more robust at avoiding local optima. They do not require a cost function gradient because this algorithm's main factors are randomness [3].

Several optimization issues are dealt with using heuristics, like the Genetic Quantum Algorithm (GQA) proposed by Han [4] and demonstrated by experimental results on the knapsack problem. Also, author Yang [5] introduces an improved ant colony optimization (IACO) for solving mobile agent routing problems, and author Kaur [6] presents a framework based on the hybridization of heuristic techniques with metaheuristic algorithm for load balancing optimization in virtual machines (VMs). Nature has been addressing complex issues for billions of years, and only the most effective and most potent solution can ensure the survival of the fittest. Similarly, heuristic algorithms solve issues by trial-and-error, learning, and adaptation. The objective is to create an efficient yet practical algorithm that will operate most of the time and produce high-quality results. It is expected that some of the quality solutions discovered will be near optimum; however, there is no guarantee of such optimal solutions. There is no certainty that the best solution will be found, and we even are not sure if an algorithm will perform or why it will if it does.

The optimization method is also still widely used by some researchers in real-world problems. Problem-solving in the real world differs entirely from problem-solving in the metaheuristic method because the problem we encounter are those we face every day and sometimes have no direct relationship with engineering. This challenge requires adaptability, endurance, ingenuity, and a certain level of creativity [7]. We frequently have to utilize multiple optimization methods for different optimization problems since not all current algorithms are possible for every particular type of problem. Therefore in this study, we propose a new approach for solving a real-world optimization issue by using MHS in

fields that do not directly address the optimization task. Indirect indicates that we provide a model problem that is conceptually similar to the problem to be solved. This model problem also assists in simplifying and become an application used to implement the MHS algorithm that we recommend. In summary, we use several known MHS and adapt the selected algorithm so that it can be used to solve optimization problems in the alternative fields we proposed and will present in this dissertation.

From the background described previously, a question emerges which we wish to prove in this study. The issues that emerge are described below.

1. Is it possible to implement the MHS search algorithms for practical problems or other than optimization?

2. Is it possible that the MHS algorithms can be easily adjusted to specific application cases?

3. Is it possible to modify the MHS algorithms for implementation in different application domains?

1.2 Research Objectives

The primary focus of this dissertation is the implementation of MHS into a new alternative field for solving complex optimization problems. We attempt to achieve this objective by researching three approaches, as shown in Figure 1.1.

Three research directions of this study are:

1. This indirectly indicates parametric study in MHS especially in Jaya algorithm.

2. Proposing the MHS into the first alternative problem in Blockchain Proof of Work (PoW).

3. Proposing the Idle-Metaheuristic into the second alternative problem in flower pollination simulation.

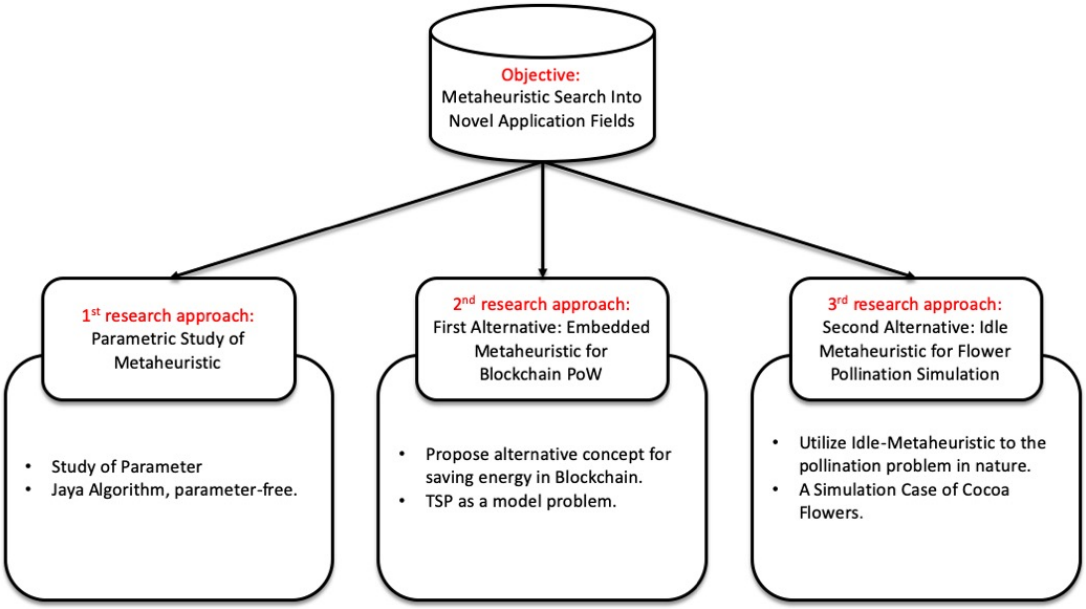


Figure 1.1: Diagram of three approacches for conducting the research.

The first research approach describes the parameter used in MHS, especially in the Jaya algorithm. We introduce some parameters and adding them to the basic Jaya algorithm to prove if the conjecture of the inventors of the algorithm, that Jaya is best used parameter-free, is true that the Jaya algorithm is parameter-free. Also, from this study, we want to know how essential the parameter is in the algorithm.

The second research approach focuses on proposing a similar scheme for generating blocks in Bitcoin Blockhain. The proposed scheme is then implemented in TSP as a problem model and uses a MHS algorithm to overcome the existing optimization problems. This scheme aims to minimize the cost of this travel, as a new block in the Blockchain, for the best n blocks to be identified by adding one more city.

The third research approach proposes an Idle-Metaheuristic that has been modified to fit the difficulties in flower pollination simulation. We investigate how three well-known yet re-targeted MHS algorithms perform when used to flower pollination problems.

In order to demonstrate the applicability of the proposed method, we used a high-level language such as MATLAB, Python and implemented it in a 3D virtual environment to demonstrate the simulation.

1.3 Chapter Section

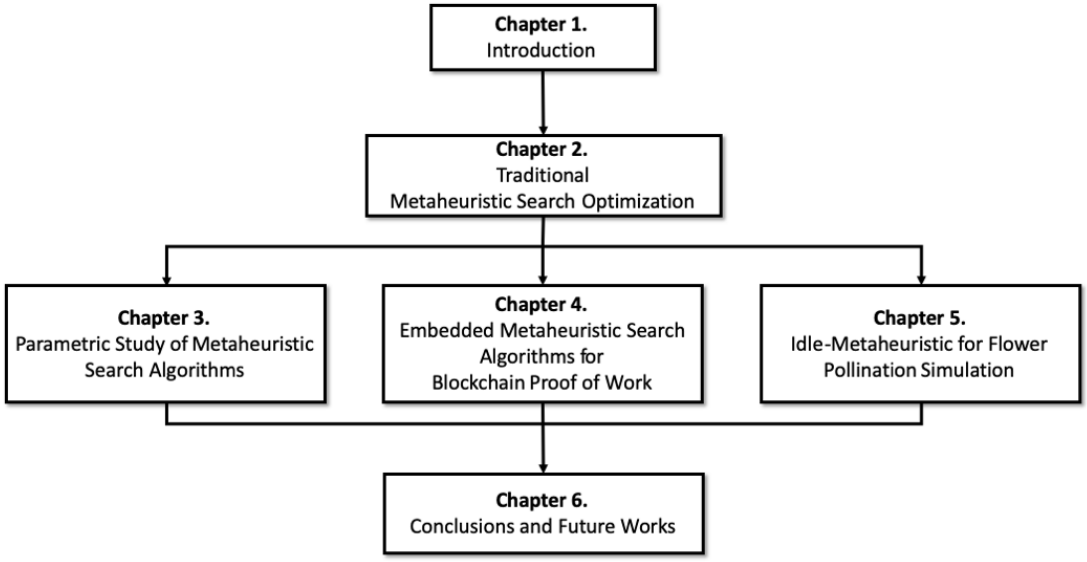


Figure 1.2: The chapter structure of this dissertation.

In this section, we describe the structure of this dissertation, as shown in Figure 1.2. Following the introductory chapter, we provided an outline of related MHS, the detailed implementation of the algorithm, and some published works in Chapter 2. We perform parametric studies on MHS and introduce weights to refer to the fact that there must be equal weights on both sides of the parameters in Chapter 3. Chapter 4 implements MHS for Blockchain Proof of Work (PoW) to reduce excessive energy consumption in bitcoin usage. In Chapter 5, we propose Idle-Metaheuristics for flower pollination simulation for behavior selection. Finally, Chapter 6 summarizes our work and the results obtained and recommends several possible research areas for future development.

Chapter 2

Traditional Metaheuristic Search Optimization

Metaheuristics search (MHS) algorithm have made much progress since the first proposals as efficient way of discovering acceptable solutions via trial and error for optimization problems that are difficult to solve exactly in practically reasonable time. MHS are now commonly used to describe all stochastic algorithms that achieve solutions through randomization and global exploration. Fred Glover explained the term ”metaheuristic” in 1986s to describe a primary approach that directs and alters other heuristics to provide solutions other than those typically obtained in the quest for local optimally [8]. There is no doubt that research in this area will continue to grow shortly, given that the term ”meta” means ”to go beyond” or ”to a higher level”. The following paragraph provide a brief history of heuristic and MHS algorithms, as shown in Figure 2.1.

Throughout the Second World War, Alan Turing was probably the first to use heuristic algorithms when he was trying to break German Enigma ciphers at Bletchley Park in 1940 [9], where Turing and British mathematician Gordon Welchman designed the Bombe, a cryptanalytic electromechanical machine, to assist their code-breaking work. Turing referred to his search method as a heuristic search because, as predicted, it worked most of the time, but there was no guarantee of finding the proper solution, but it was an enormous success.

Then, in the 1960s and 1970s, John Holland and his colleagues at the University of

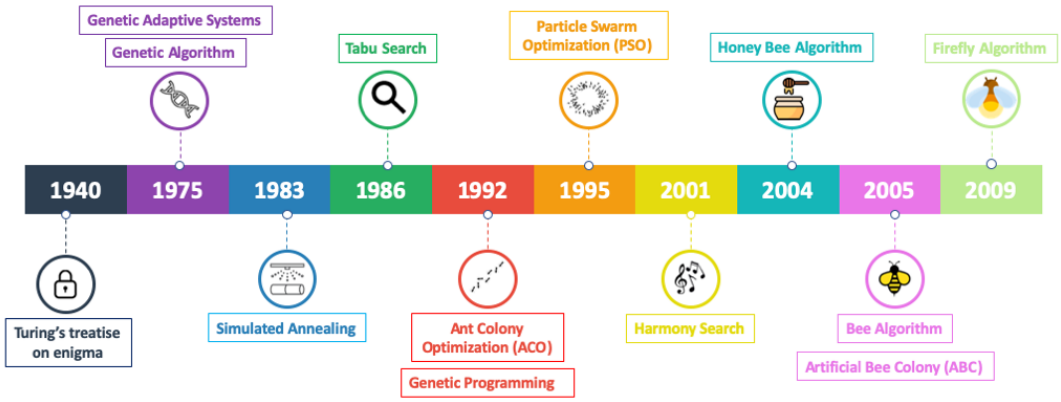


Figure 2.1: A brief history of heuristic and metaheuristic optimization.

Michigan were the first to create evolutionary algorithms. Holland investigated adaptive systems and was the first to use crossover and recombination manipulations to simulate such systems. His seminal work, which included a description of Genetic Algorithms, was released in 1975 [10]. During the same year, Kenneth De Jong [11] published his dissertation on the potential and power of Genetic Algorithms for several objective functions, whether noisy, multimodal, or even intermittent.

With the advent of a crucial step in the development of Simulated Annealing (SA) [12], an optimization method pioneered inspired by the metal annealing process, the 1980s and 1990s became exciting years for MHS algorithms. It is a trajectory-based search technique that begins with an initially assumed solution at a high temperature and gradually cools the system down. The first actual use of MHS was apparently due to Tabu Search in 1986, even though the seminal book on Taboos was published later in 1997 [13].

Marco Dorigo released his seminal work on Ant Colony Optimization (ACO) in 1992 [14], inspired by the intelligence of social ant swarms that use pheromones as chemical messengers. In the same year, John R. Koza [15] offered a treatise on Genetic Programming, laying the groundwork for a new field of machine learning that would transform

computer programming.

Shortly afterward, in 1995, American social psychologist James Kennedy and engineer Russell C. Eberhart created Particle Swarm Optimization [16], which was inspired by the intelligence of swarms of fish and birds and human behavior. There have been around 20 different particle swarm variations developed since its inception, and it has been used in practically all fields of challenging optimization problems.

Things were getting more intriguing around the turn of the century with the release of the Harmony Search (HS) [17] method in 2001, which has been widely used in tackling numerous optimization issues such as water distribution, transportation modeling, and scheduling. Following the Honey Bee algorithm [18], which was used to enhance Internet hosting centers in 2004, came the invention of a new bee algorithm by [19] and Artificial Bee Colony (ABC) [20] in 2005. The Firefly Algorithm [21] was then published in 2009, inspired by the behavior of fireflies and flashing patterns.

More and more MHS algorithms are being designed, and many researchers are still interested in researching this method. Various algorithms necessitate a system summary of various MHS algorithms, and this dissertation aims to present MHS and their applications to a new field. One of the reasons why MHS are still in high demand among academics is because they feature the following characteristics:

- Various trade-offs are using each MHS algorithm to choose between randomization and local search. MHS algorithms solve optimization issues by finding reasonable solutions to complicated problems in reasonable periods, but there is no assurance that optimum solutions will always be discovered. Almost all MHS algorithms are well-suited to global optimization.

- Diversification and intensification (or exploitation and exploration) are the two most essential components of all MHS algorithms [22, 23]. Diversification involves algorithms that try to generate new solutions to explore the search space globally, whereas intensification algorithms focus on a specific location, knowing that great solutions are presently discovering in this region. When finding the optimal solution, a proper balance between diversity and intensification must be created to enhance the algorithm's convergence speed. A good combination of these two main components

nearly usually assures that a global optimum can be achieved.

- Nature is one characteristic that significantly impacts MHS algorithms that explore search space with a population or single solution. Local search-based MHS such as Tabu Search and Simulation Annealing are examples of methods that employ a single solution. These search method have a characteristic that defines the state of the search space during the search process. On the other hand, population-based MHS investigate the search space by generating a series of solutions in the search space, such as Genetic Algorithms.

Developing a MHS algorithm aims to solve complex optimization problems when previous optimization algorithms have failed to find the optimum. Currently, these methods are regarded as some of the most practical options for dealing with a wide range of real-world issues [24]. The use of MHS algorithms for optimization has several advantages, such as:

1. MHS algorithms have a wide range of applications since they could be used to any problem represented as a function optimization problem. For example, in the case of Genetic Algorithms, we only need to code potential solutions, but in principle, we could use Genetic Algorithms to a variety of problems, but they will not always be the optimal solution for each of these difficulties. Furthermore, unlike gradient-based optimization method, the gradient of the objective function is unnecessary, like in the case of Genetic Algorithms, all that is required in evaluating the solution (for example, suitability or novelty).

2. In order to avoid local minima, MHS algorithms frequently use some randomization. ACO algorithms and Simulated Annealing exemplify this technique.

3. MHS algorithms can also be combined with more traditional optimization techniques or other optimization techniques, as in [2], which proposes a new MHS algorithm based on the exciting breeding behavior of certain species of cuckoos, such as brood parasitism, and combines it with Lévy Flight. Furthermore, in [25], a combination of Artificial Neural Network (ANN) and Particle Swarm Optimization (PSO) is used for predicting the cemented paste backfill.

4. Generally, MHS algorithms are easier to comprehend and implement.

5. MHS algorithms' efficiency and adaptability enable them to solve more significant problems more rapidly.

Aside from its advantages, the MHS algorithm have some disadvantages too, which are as follows:

1. The performance of optimization is highly dependent on precise parameter adjustment.

2. The MHS algorithm cannot guarantee optimality.

3. The MHS algorithm does not appear to be able to reduce the search space.

4. It is not guaranteed that optimization results acquired with the same initial condition settings will be repeatable.

Figure 2.2 shows the MHS classification, as stated in the MHS characteristics, and in the following section, we present most typical and well-known algorithms used in single-solution and population-based method.

2.1 Single Solution-Based Metaheuristic

Single solution-based metaheuristic are ones in which a solution is produced at random and improved until the best result are achieved. Single solution method concentrate on changing and enhancing a single potential solution. Simulation Annealing (SA) and Tabu Search (TS) are two well-known MHS that use single solution search. Since it only reforms one solution generated at random for the given problem, the search procedure used by this approach could be trapped in local optima, preventing us from finding the global optimum [26]. However, the solution (new state) created by this method may not be derived from the present solution environment (state) [27].

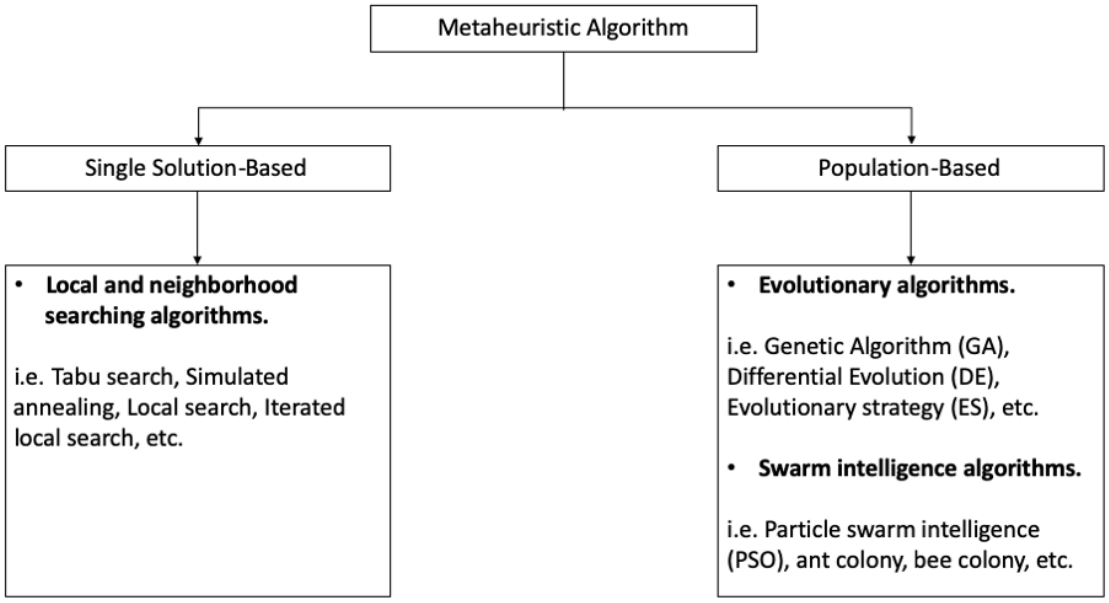


Figure 2.2: Metaheuristic classification.

2.1.1 Simulated Annealing (SA)

Simulate Annealing algorithm (SA) was developed first in 1953, and Kirkpatrick [12] used it as a search algorithm for combinatorial optimization problems. This method is motivated by the physical annealing process, slowly heating and cooling materials to achieve a uniform structure.

SA is a stochastic method for avoiding non-global local minima while exploring global minima. During the minimization process, the chances of deterioration decrease towards zero. This disruption enables to transcend the local minima and thoroughly explore the searchspace (s). As a result, this approach will reach a global minimum [28].

According to the pseudocode of SA 2.1 [29], the searchspace (s) is the probability of transitioning from the current state to the new state s_{new}, as indicated by the acceptance probability function $P(e, e_{new}), T)$. This probability is determined by the energies $e = E(s)$ and e_{new} $E(s_{new})$ of the two states, as well as the global time-varying parameter T known as temperature. States with lower energy levels perform better than states with higher energy levels. Even though e_{new} is more significant than e, the probability function P must be

Algorithm 2.1: Simulated Annealing

1: **Let** $s = s_0$
2: **For** $k = 0$ through k_{max} (exclusive):
3: $T \leftarrow (1 - (k + 1)/k_{max})$
4: Pick a random neighbour, $s_{new} \leftarrow neighbour(s)$
5: **if** $P(E(s), E(s_{new}), T) \geq random(0.1)$:
6: $S \leftarrow s_{new}$
7: **end if**
8: **end for**
9: Output: the first final state S

positive. This process prevents the algorithm from getting stuck on a local minimum which is worse than the global minimum.

In SA, solution optimization requires neighboring assessments of neighboring states, which are new states generated by conservatively changing certain states. Like the TSP, each state is often described as a permutation of the cities to be toured, and each state's neighbors are the set of permutations created by exchanging these two cities. A "move" is a well-defined technique of altering the states to generate neighboring states, and various moves yield different sets of neighboring states. These moves typically result in minor changes to the previous state to incrementally enhance the solution by iteratively enhancing its components (as in the traveling salesman problem's city connections).

Modest heuristics like hill-climbing [30], which move by finding a better neighbor after a better neighbor and stop when they find a solution with no neighbors that are better solutions, cannot guarantee that they will lead to any of the current better solutions; their outcome could easily be a local optimum. Meanwhile, the actual best solution would be a global optimum that could be different. MHS utilize solutions from neighbors to explore the solution space. Although they prefer to find a more suitable neighbor, at the same time, they also will tolerate a worse neighbor in order to avoid the problem of being stuck in the local optima so that when run for an extended period, they can discover the global optimum.

2.1.2 Tabu Search

Tabu Search (TS) is a MHS search method that utilizes local search methods to solve complicated challenges in mathematical optimization. This method is also known as Adaptive Memory Programming, developed by Fred W. Glover in 1986 [8].

According to Xin-She [31], the first actual implementation of MHS was most likely due to Fred Glover's search for TS in 1986, even though a seminal book on the research for TS was released later in 1997.

Although the TS was introduced decades ago, it still provides solutions that are pretty near to optimality and are among the most successful, if not the best, in overcoming the challenging problems encountered. This success has made TS highly popular among researchers interested in discovering better solutions, and several papers, book chapters, special editions, and books have studied the enormous TS field [32, 33, 34, 35].

Algorithm 2.2: Tabu Searcch

```
1:  sBest ← s0
2:  bestCandidate ← s0
3:  tabuList ← []
4:  tabuList.push(s0)
5:  while (not stoppingCondition())
6:      sNeighborhood ← getNeighbors(bestCandidate)
7:      bestCandidate ← sNeighborhood [0]
8:      for (sCandidate in sNeighborhood)
9:          if ((not tabuList.contains(sCandidate)) and
    (fitness(sCandidate) > fitness(bestCandidate))) bestCandidate ← sCandidate
10:         end
11:     end
12:     if (fitnes(bestCandidate) > fitness(sBest))
13:         sBest ← bestCandidate
14:     end
15:     tabuList.push(bestCandidate)
16:     if (tabuList.size > maxTabuSize)
17:     tabuList.removeFirst()
18:     end
19: end
20: return sBest
```

The pseudocode 2.2 [36] at lines (1-4) indicates some initial setup, which includes constructing an initial solution (probably selected at random), marking that first solution as the best seen to date, and initializing a Tabu list using this initial solution. The primary algorithmic loop begins on line 5 and continues to find the best solution until the predefined termination criteria are reached (two examples of these conditions are simple time limits or thresholds on fitness scores). Line (9) searches for Tabu components in neighboring solutions. Furthermore, the algorithm searches for the optimal solution in the environment, which is not Tabu (means not in the Tabu list yet). If the best local candidate has a better fitness value than the existing best in line (12), the new best candidate is assigned to line (13). Line (15) always includes the local best candidate to the Tabu list, and if the Tabu list is complete in line (16), some components are allowed to expire in line (17). In general, components leave the list in the same order they were inserted. In order to avoid the local optimum, the process will choose the best local candidate (even if it has lower fitness than the $sBest$). This procedure is repeated until the user-specified termination condition is reached, at which time the best solution found throughout the search is returned in line (21).

The pseudocode 2.2 implementation provides undeveloped short-term memory but does not contain medium or long-term memory structures. The statement ”fitness” refers to a measure of the candidate solution as represented by an objective function for mathematical optimization.

One method of utilizing the search history is to use the Tabu list as short-term memory. The information obtained can be added to TS during the search process using four distinct principles: frequency, recency, quality, and influence. Frequency-based memory takes advantage of the number of visits to each solution attribute. The participation of each solution attribute in the current iteration correlates to recency-based memory. This information can be used to discover the search process limitations and the necessity for diversification in the search process. Solution quality can be used to discover better building blocks and direct the search process. The impact attribute relates to determining which solution attributes are essential in directing the search process and may thus be given some preference throughout the search operation. Robust TS [37] and reactive TS [38] are several of the TS algorithms that use these four different principles. To summarize, TS continues to be a significant

source of inspiration and strategies used by various MHS.

2.2 Population-Based Metaheuristic

A population-based metaheuristic directs the search process by retaining several solutions at various points in the search space. The population-based metaheuristic in the search space can sustain the diversity of the solution globally with multiple starting points, resulting in greater exploration during the search process.

In contrast to single-based metaheuristics, population-based metaheuristics begin with a group of solutions (population). They then construct a new solution population iteratively. Information can be transferred between the set of solutions in this way. The main benefit of population-based metaheuristics is that they avoid getting stuck in local optima. This kind of MHS algorithm is one of the most well-known optimization algorithms, and it has been widely used in different areas, including medical systems [39, 40], automotive engine design [41], and quality of food [42].

Evolutionary computers [43], Genetic Algorithms [44], and Particle Swarm Optimization [16] are examples of population-based metaheuristics. Moreover, population-based metaheuristics include swarm intelligence, as collective behavior of decentralized and self-organized individuals in a population or swarm. Examples in Swarm Intelligence include Ant Colony Optimization [45], Particle Swarm Optimization, and Social Cognitive Optimization [46]. In the next section, we describe some of the most well-known population-based metaheuristics and others relevant to our study.

2.2.1 Genetic Algorithm

John Holland [47] proposed the Genetic Algorithm (GA), inspired by Charles Darwin's theory of natural evolution and based on biological evolution concepts. This algorithm represents the natural selection process, in which the best appropriate individuals are chosen for reproduction to generate descendants from the following generation.

Natural selection begins with the selection of the most appropriate individuals from a population. They have descendants that inherit the features of their parents and will be

handed to the next generation. Parents who have better fitness will produce better descendants, and these descendants will have a higher chance of surviving. This procedure keeps being repeated until the generation with the most suited individuals is discovered. This natural selection process can be divided into five steps: initial population, fitness function, selection, crossover, and mutation.

Algorithm 2.3: Genetic Algorithm

```
1:  begin
2:      t = 0
3:      Create an initial population - Pop(0)
4:      Evaluate individuals - calculate the value of the fitness function for each individual
5:      in the population P_0
6:      do
7:          select individuals for the new population Pop(t) - selection
8:          perform crossover operation
9:          perform the mutation of individuals
10:         evaluate individuals
11:         replace old population a new one
12:         t = t + 1
13:         while stop condition reached
14: end
```

GA will stop the iterations when the conditions are reached, as shown in the pseudocode 2.3 [48] above. This situation indicates that the population has converged means does not generate descendants significantly different from the previous. The final result of this descent is the best solution to the problem that GA is solving. GA has been widely used to multiple optimization issues such as in engineering [49] and chemistry [50] and has recently been expanded to data mining and machine learning technologies [51] as well as the rapidly expanding field of bioinformatics [52].

2.2.2 Ant Colony Optimization

The Ant Colony Optimization algorithm (ACO) [53] is a MHS method for solving discrete optimization problems by finding a suitable path through graphs. ACO algorithm is essentially an artificial ant that stands for a multi-agent method inspired by ant behavior in which

agents collaborate through low-level interactions.

ACO's fundamental paradigm is based on ant colony foraging behavior. Ants release pheromones in the soil throughout their travel. The ants most likely follow the pheromones left behind by the previous ants. Local ants can determine the shortest path from the food source to the nest this way. This procedure is then adopted and used by the ACO algorithm to perform optimization.

Algorithm 2.4: Ant Ccolony Optimization

```
1:  Begin
2:      Initialize
3:      While terminated criterion not satisfied do
4:          Position each ant in a starting node
5:          Repeat
6:              For each ant do
7:                  choose nect node by applying the state transition rule
8:                  Apply step by step pheromone update
9:              End for
10:         Until every ant has built a solution
11:         Update best solution
12:         Apply offline pheromone update
13:     End While
14: End
```

From the pseudocode 2.4 [54], an example of a global pheromone updating rule is

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau_{xy}^{k} \tag{2.1}$$

Notation:

τ_{xy}	The amount of pheromone deposited for a state transition x y;
ρ	The pheromone evaporation coefficient;
m	The number of ants and;
$\Delta\tau_{xy}^{k}$	The amount of pheromone deposited by k .

An artificial ant is an essential computational agent that searches for reasonable solutions to an ACO algorithm optimization problem. The optimization issue must be transformed into the challenge of finding the shortest path on a weighted graph before using an ACO algorithm. Each ant stochastically generates a solution, i.e., the order in which the edges in the graph should be followed, in the initial step of each iteration. The pathways discovered by the several ants are compared in the second phase. The final step is to update the pheromone levels on each edge.

2.2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is one of the MHS method introduced by Kennedy [16] and Shi [55] that replicates social behavior as a stylized depiction of the movement of animals in a flock of birds or fish school.

The typical PSO method employs a swarm of particles or a population of possible solutions that move in the search space based on their position and velocity. Its best-known position directs the particle's movement and the best-known position of the entire swarm to move all particles to the global best solution in the solution space. The swarm's best-known position is updated at each iteration. This procedure is performed for a certain number of iterations.

$$V_i(i+1) = W \times V_i(t) + C_1 \times rand() \times (\mathbf{pbest} - X_i) + C_2 \times rand() \times (\mathbf{gbest} - X_i) \quad (2.2)$$

$$X_i(t+1) = X_i(t) + V_i(i+1) \quad (2.3)$$

Notation:

$V_i(i+1)$ and $X_i(t+1)$	The velocity and position of $i-th$ particle in the t generation;
pbest	The past best position of $i-th$ particle;
gbest	The best faund position so far;
rand()	Generates a random number between (0,1);
W	The inertia factor;
C_1 and C_2	Learning factors and they are usually set to constants.

Algorithm 2.5: Particle Swarm Optimization

```
1: Initialization the velocity and position of the population randomly.
2: Evaluate the population.
3: while the termination condition is not satisfied do.
4:     for i = 1 to population size do.
5:         Update the velocity of i − th particle using the Eq. (2.2).
6:         Update the position of i − th particle using the Eq. (2.3)
7:         Evaluate the i − th particle.
8:     end for
9: end while
10: Output the found optimum
```

The PSO algorithm 2.5 [56] has received much attention due to its simplicity and ease of use with few parameters; therefore, many researchers are still willing to develop a better variation of PSO. Several researchers investigated the effect of parameters on PSO performance [57, 58]. Other researchers implemented a variety of novel techniques by combining PSO with other algorithms [25, 59, 60]. Furthermore, PSO has been adjusted too following the problem to be optimized [61, 62].

2.2.4 Differential Evolution

In evolutionary computing, Differential Evolution (DE) is a population-based optimization technique that Storn and Price [63] proposed to solve a real-valued continuous optimization problem.

DE optimizes the problem by iteratively attempting to improve candidate solutions concerning a particular quality measure. These approaches are commonly referred to as MHS

since they make few or no assumptions about the optimized problem and explore many potential solutions. On the other hand, MHS like DE do not ensure that an optimal solution is ever discovered.

Algorithm 2.6: Differential Evolution

```
Initialization population randomly.
Evaluate the population.
while the termination condition is not satisfied do.
    for i = 1 to population size do.
        Select an individual randomly as a base vector, X_base.
        Select two individual randomly to construct a difference vector, (X_r1 − X_r2,
        where r1 ≠ r2 ≠ base ≠ i.
        Generate a mutation vector, X_mutation = X_base + scalingfactor × (X_r1 − X_r2).
        Crossover X_mutation and X_i to generate a trial vector, X_trial.
        Evaluate the fitness of the trial vector, X_trial.
        The winner of X_trial and X_i can survive to the next generation.
    end for
end while
Output the found optimum
```

The primary DE method utilizes a population of randomly initialized solutions in the search space as a set of agents. Each agent is relocated in each iteration depending on a combination of other agents in the population using the differential weight parameters and crossover probability. The algorithm executes for a certain number of iterations.

DE is one of the most widely used algorithms in the field of evolutionary algorithms. Many efficient alternatives have been proposed, including combinatorial optimization [64, 65], clustering [66], electromagnetics [67], and others [68, 69, 70].

2.3 Summary

In this chapter, we provide readers with a fundamental understanding of MHS. Then, we introduced some well-known algorithms, along with demonstrations of their implementation in various fields. Finally, to solve the problem in this study, we chose several algorithms that focus on population-based MHS, such as the Jaya Algorithm, Particle Swarm

Optimization (PSO), Quantum Behave PSO (QPSO), Crow Search Algorithm (CSA), and Cuckoo Search, which then adjusts the problem to be solved.

Chapter 3

Parametric Study of Metaheuristic Search Algorithms

3.1 Study of Jaya Algorithm

3.1.1 Introduction

Rao introduced the Jaya algorithm as a swarm optimization method in 2016 [71]. The Jaya method is well-known for a parameter-free and simple algorithm with the idea that the solution found for a given problem moves toward the best solution and away from the worst solution. Rao proposed the Jaya method to solve constrained and unconstrained optimization problems in his research. This method claims to be parameter-free and modest because it just requires the standard control parameters and no algorithm-specific control parameters.

In 2011, Rao introduced the TLBO (teaching-learning-based optimization) algorithm [72]. Many researchers are also interested in this algorithm since it introduces a learning-based optimization technique requiring no algorithm-specific parameters. Following the popularity of the TLBO algorithm, Rao then proposes the Jaya algorithm that is claimed to be free of other particular parameters and operates differently. The TLBO algorithm consists of two phases: the teacher phase and the student phase, whereas the Jaya algorithm has just one phase (shown in Figure 3.1) and is relatively easy to implement. In his article

[71], Rao compares the Jaya algorithm with the TLBO algorithm and includes several other well-known MHS algorithms (PSO, DE, GA, ABC, etc.). The performance of Jaya's algorithm and other algorithms was evaluated using 24 benchmark functions of CEC 2006 (also 30 unconstrained benchmark functions) in addition to statistical tests such as the Friedman ranking test [73] and the Holm-Sidak test [74] is used to prove the significance of Jaya's algorithm and other algorithms. The findings indicate that the Jaya algorithm performs well, and it is concluded that the Jaya algorithm can be utilized to solve both constrained and unconstrained optimization problems. We cannot claim that Jaya's algorithm is the "best" algorithm when compared to others, but what researcher Rao wants to highlight about Jaya's algorithm is that it is simple to implement, has no algorithm-specific parameters, and generates optimal results in a relatively small number of function evaluations.

The Jaya algorithm's structure is quite similar to that of Particle Swarm Optimization (PSO). The particles in PSO work towards both the personal best and the global best simultaneously; meanwhile, the Jaya algorithm prefers to lead to the best solution and away from the worst solution and has no inertia. As shown in the following equation:

$$X_j^{(i+1)} = X_i + r_1 \cdot (X_{b,j} - |X_j^i|) - r_2 \cdot (X_{w,j} - |X_j^i|) \tag{3.1}$$

Notation:

$X_j^{(i+1)}$	The update value of X_j^i and r_1 the best candidate;
$X_{b,j}$	The value of the variable j for the best candidate;
$X_{w,j}$	The value of the variable j for the worst candidate;
r_1 and r_2	Random numbers i.i.d. from [0.1].

It is described in formula 3.1 that $X_j^{(i+1)}$ is the updated value that has met the requirements of being satisfied or the best candidate. To obtain the condition satisfied value, $X_{b,j}$ includes the best candidate from the value of a variable j in all candidate solutions, whereas $X_{w,j}$ contains the worst candidate from variable j in all candidate solutions. Then

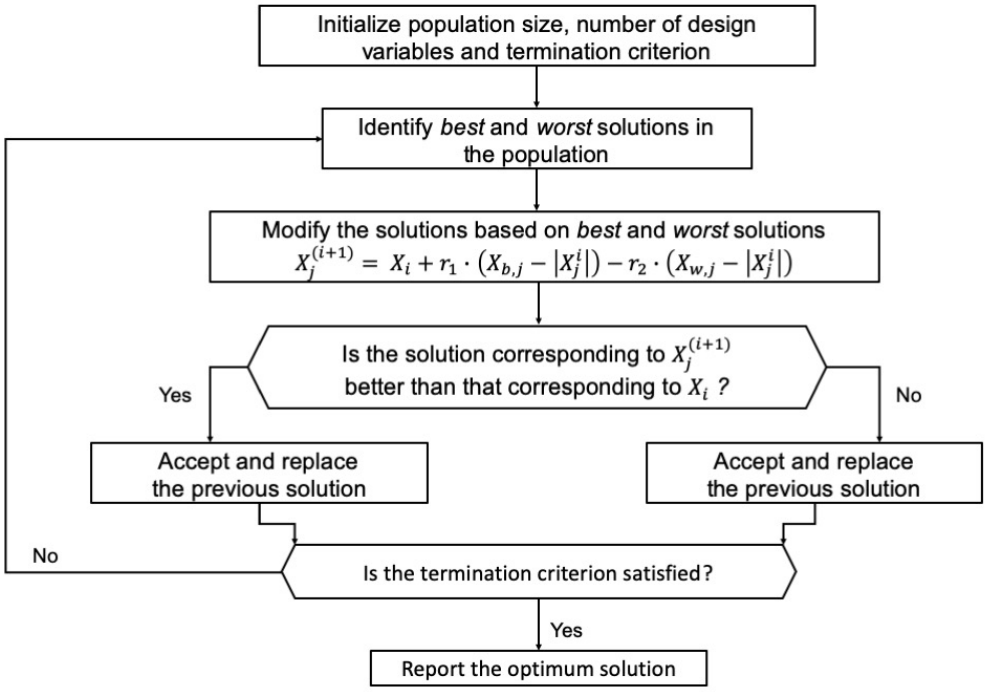


Figure 3.1: Flowchart of the Jaya algorithm.

for "$r_1 \cdot (X_{b,j} - |X_j^i|)$" represents the solution's propensity to get closer to the best solution, and "$-r_2 \cdot (X_{w,j} - |X_j^i|)$" represents the solution propensity to avoid the worst solution. In conclusion, the best and worst solutions are generated by updating the iteration values of the distance tendencies of the best and worst candidate solutions, which are combined along with the initial and random values.

When a position on *Xbest* and *Xworst* changes and has a better objective value, it will replace the current status. The Jaya equation can also describe as the previous position plus the best term minus the worst term. Another distinction between Jaya and PSO is that in Jaya, the best and worst solutions will update at each time of iteration. Still, in PSO, the global best and personal best are updated whenever a better solution is identified [75]. Figure 3.1 describes the Jaya algorithm flowchart.

After Rao presented the Jaya algorithm, it quickly gained popularity among some researchers and became one of the most potent ways for solving various issues. Kunjie Yu [76] introduced the performance-guided Jaya algorithm (PGJAYA) to assess, regulate, and

track maximum power points in photovoltaic (PV) systems. The PGJAYA method is performed to extract parameters from various PV models. As a result of its improved performance at different irradiation and temperatures, PGJAYA has emerged as a viable parameter identification method for PV cell and module models. Moreover, the Jaya algorithm has also been modified by using other algorithms to increase Jaya's performance. This research [77] overcomes the problem of being stuck in local minima by combining Levy Flight and Greedy selection concepts into Jaya's basic algorithm.

Most existing swarm-based algorithms include standard controlling parameters, such as particle number, or specific parameters, such as weight and global best in the PSO algorithm shown in the equation (2.2) and (2.3) described in Chapter 2. In addition, the equation (3.2) from the Firefly algorithm [78] also shows some of the parameters used, such as β, α, and ϵ. Meanwhile, the Jaya equation (3.1) only consists of random values, the best and the worst positions. Since the Jaya algorithm requires only the standard control parameters and does not require any algorithm-specific control parameters, tuning is not required. Therefore in this work, we want to investigate the influence of new parameter settings on the Jaya algorithm by doing a parametric study. This method is evaluated on seventeen unimodal and multimodal benchmark functions [79], and the experimental results showed considerable improvements in exploration capabilities and convergence speed without being trapped in local optima.

$$X_i^{t+1} = X_i^t + \beta_0 e^{-\gamma r_{ij}^2}(X_j^t - X_i^t) + \alpha_t \epsilon_i^t \tag{3.2}$$

Notation:

- β_0 The attractiveness at $r = 0$.
- α_t The randomization parameter.
- ϵ_i^t Vector of random numbers drawn from a Gaussian distribution or uniform distribution at time t.

3.1.2 Some Insights Into Jaya Algorithm

It has previously been stated several times that the Jaya method is parameter-free. There has not been any investigation so far to see if that is true, indicating method to improve Jaya by incorporating parameters hidden in the original formulation. This section present a theoretical explanation for why the usage of such hidden parameters can be advantageous.

Suppose an objective function best described by the metaphor "island in the middle of a lake." What we mean is a real-valued function defined over $\mathbf{R}$ and two range values a and b. For $|x| > a$ we set $f(x) = 1$, for $a \geq |x| \geq b$ we set $f(x) = 1000$ and for $|x| < b$ we set $f(x) = 2$. Concerning the metaphor, the first case is the mainland, providing the absolute minimum of the function, the second part the lake, with worse objective values, and surrounding the island of the third case, technically a local optimum. Furthermore, in this case, we are considering the minimization problem.

Regarding the Jaya algorithm update formula, Eq. (3.1) it can be observed that the magnitude of change is within intra-population distances. The change in the x position can be at most the largest difference between any coordinate of any individual. But then, the modified x-position is only updated if there is an improvement in the objective function values (compared to PSO, Jaya doesn't have inertia). It means if we choose a small b (small island, say $b = 1$) and large a (far away from mainland, say $a = 100$) and also assuming that the initialization of Jaya happened such that all individuals are located on the island, there will be never a probe of a position far enough from the island to reach the mainland. In all cases it will be $|x| \leq 2$ and $f(x)$ either 2 or 1000, no x will ever reach an optimum position with $|x| > a$. Thus, Jaya will become stuck on the island.

Therefore in case, introducing parameters might assist, be it even a theoretical and impractical way. For example, consider the case where weights are added in Eq. (3.1) for the repulsion term. If we select a weight large enough, it can be sufficient to probe x values of $|x| > a$. Of course, we can also set a range parameter for r_1, r_2 with same effect. This is a purely theoretical argument, and we do not know how it will impact Jaya's effectiveness when dealing with real-world problems or test functions. A series of experiments described below were carried out to determine the impact of weighting parameters on performance.

The second insight is about the deviation of Jaya Eq. (3.1) from a pure vector notation.

We mean the use of $|X_j^i|$ in the Jaya update rule instead of just X_j^i. The latter case would describe a vector pointing away from the worst or towards the best. Using the absolute, and once there is a mixture of positive and negative component values, the change of an individual vector becomes a somewhat unpredictable issue. So far, we could not find any explanation or consideration about the choice of the absolute value, but given Jaya's result on various applications from literature, it does not seem to be a drawback. Therefore, it is also interesting to see what happens if we select other functions instead of the absolute value.

In the next section, we conduct a series of experiments to investigate the influence of hidden parameter settings on Jaya performance, i.e.:

- weights for the first and second update term,
- ordered-weights for also taking second-worst and second-best into account, and
- variants of the coordinate function.

3.2 Methodology

3.2.1 Proposed Concept

The effectiveness of optimization is closely related to the fine-tuning of all parameters. Some swarm-based algorithms have general controlling parameters, such as the number of particles, while others, such as the PSO algorithm, have specific controlling parameters, such as weights and global best. Meanwhile, Jaya's method requires general control parameters and does not require algorithm-specific control parameters; thus, tuning is not required. Therefore the effect of setting new parameters on the Jaya method will be introduced in this section.

This concept is based on our objective of figuring out how the Jaya equations may be simple while still producing efficient results based on the best and worst solutions at the same time. More precisely, which of the two parts of the optimization process is the most efficient. Therefore, in conclusion, we introduce weights to indicate that both the best and worst sides must have equal weights. This argument is premised on the study's goal of

confirming whether Jaya's algorithm is parameter-free; therefore, we propose giving equal weight to the distance to the best and the distance away from the worst. Following that, determining which of the second best and second worst impacts is more influential offers a positive impression of Jaya's algorithm's performance.

We investigated two items to see how hidden parameter settings affected Jaya's performance: The first is to change the weights for best and worst, and the second is to change the weights by incorporating second best and second-worst. Here, we examined the performance of Jaya's algorithm by implementing 12 unconstrained Benchmark functions to obtain concrete results that show the influence of parameter settings on the performance of Jaya's method.

1. The first test uses two distinct weights (1 0 2) and (2 0 1). Then make a comparison to the results of the Jaya algorithm. The number 0 in the middle represents the solution center group; the first weight is for the best, and the third is for the worst, other than that is 0. From the original equation 3.1, we can derive Jaya's formula as follows:

$$X_j^{(i+1)} = X_j^{(i)} + 1 \cdot r_1 \cdot (X_{b,j} - |X_j^{(i)}|) - 2 \cdot r_2 \cdot (X_{w,j} - |X_j^{(i)}|) \tag{3.3}$$

$$X_j^{(i+1)} = X_j^{(i)} + 2 \cdot r_1 \cdot (X_{b,j} - |X_j^{(i)}|) - 1 \cdot r_2 \cdot (X_{w,j} - |X_j^{(i)}|) \tag{3.4}$$

2. We utilized five additional weights for the second test: (0.9 0.1 0 0.1 0.9), (0.7 0.3 0 0.3 0.7), (0.5 0.5 0 0.5 0.5), (0.3 0.5 0 0.5 0.3), and (0.8 0 0 0 0.8). In terms of notation, the same applies here, but weights for second best and worst are introduced as follows:

$$\begin{aligned} X_j^{(i+1)} = X_j^{(i)} &+ 0.9 \cdot r_1 \cdot (X_{b1,j} - |X_j^{(i)}|) + 0.1 \cdot r_2 \cdot (X_{b2,j} - |X_j^{(i)}|) \\ &-0.1 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) - 0.9 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) \end{aligned} \tag{3.5}$$

$$X_j^{(i+1)} = X_j^{(i)} + 0.7 \cdot r_1 \cdot (X_{b1,j} - |X_j^{(i)}|) + 0.3 \cdot r_2 \cdot (X_{b2,j} - |X_j^{(i)}|) \\ -0.3 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) - 0.7 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) \tag{3.6}$$

$$X_j^{(i+1)} = X_j^{(i)} + 0.5 \cdot r_1 \cdot (X_{b1,j} - |X_j^{(i)}|) + 0.5 \cdot r_2 \cdot (X_{b2,j} - |X_j^{(i)}|) \\ -0.5 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) - 0.5 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) \tag{3.7}$$

$$X_j^{(i+1)} = X_j^{(i)} + 0.3 \cdot r_1 \cdot (X_{b1,j} - |X_j^{(i)}|) + 0.5 \cdot r_2 \cdot (X_{b2,j} - |X_j^{(i)}|) \\ -0.5 \cdot r_1 \cdot (X_{w1,j} - |X_j^{(i)}|) - 0.3 \cdot r_2 \cdot (X_{w1,j} - |X_j^{(i)}|) \tag{3.8}$$

$$X_j^{(i+1)} = X_j^{(i)} + 0.8 \cdot r_1 \cdot (X_{b,j} - |X_j^{(i)}|) - 0.8 \cdot r_2 \cdot (X_{w,j} - |X_j^{(i)}|) \tag{3.9}$$

3.2.2 Unconstrained Benchmark Functions

A test function is required to validate and compare Jaya's performance with additional weight parameters. Many tests or benchmark functions have been widely utilized for evaluating new algorithms, and in this study, we choose 12 unconstrained Benchmark functions, as provided in Table 3.1 from the literature survey of benchmark function [79].

In the following, we briefly explain 12 unconstrained optimization test problems that can be used to validate the performance of optimization algorithms.

- **Sphere Function** [80] (Continuous, Differentiable, Separable, Scalable, Multimodal)

$$f_{Sphere}(x) = \sum_{i=1}^{D} x_i^2 \tag{3.10}$$

Table 3.1: Unconstrained benchmark functions.

F	Function	Dimension	Search Range	C
F1	Sphere	30	[-100,100]	MS
F2	Sum Squares	30	[-10,10]	US
F3	Beale	5	[-4.5, 4.5]	UN
F4	Easom	2	[-100, -100]	MS
F5	Matyas	2	[-10,10]	UN
F6	Colville	4	[-10,10]	MN
F7	Zakharov	10	[-5, 10]	MN
F8	Rosenbrock	30	[-30,30]	UN
F9	Branin	2	[-5,10]	MN
F10	Booth	2	[-10,10]	UN
F11	GoldStein-Price	2	[-2,2]	MN
F12	Ackley	30	[-32,32]	MN

Note: F: Function, C: Characteristic, MS: Multimodal Separable, US: Unimodal Separable, UN: Unimodal Non-separable MN: Multimodal Non-separable.

subject to $0 \leq xi \leq 100$. The global minimum is located at $x^* = f(0, ..., 0), f(x^*) = 0$

- **Sum Squares Function** [81] (Continuous, Differentiable, Separable, Scalable, Unimodal).

$$f_{SumSquares}(x) = \sum_{i=1}^{D} ix_i^2 \tag{3.11}$$

subject to $-10 \leq xi \leq 10$. The global minima is located $x^* = f(0, ..., 0), f(x^*) = 0$.

- **Beale Function** [79] (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal).

$$f_{Beale}(x) = (1.5 - x_1 + x_1x2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1X_2^3)^2 \tag{3.12}$$

subject to $-4.5 \leq xi \leq 4.5$. The global minimum is located at $x^* = (3, 0.5), f(x^*) = 0$.

- **Easom Function** [82] (Continuous, Differentiable, Separable, Non-Scalable, Multimodal).

$$f_{Easom}(x) = -\cos(x_1 \cos(x_2) \exp\left[-(x_1 - \pi)^2 - (x_2 - \pi)^2\right] \tag{3.13}$$

subject to $-100 \leq xi \leq 100$. The global minimum is located at $x^* = f(\pi, \pi), f(x^*) = -1$.

- **Matyas Function** [81] (Continuous, Differentiable, Non-Separable, Non-Scalable, Unimodal).

$$f_{Matyas}(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \tag{3.14}$$

subject to $-10 \leq xi \leq 10$. The global minimum is located at $x^* f(0, 0), f(x^*) = 0$.

- **Colville Function** [79] (Continuous, Differentiable, Non-Separable, Non-Scalable, Multimodal).

$$\begin{aligned} f_{Colville}(x) = 100(x_1 - x_2^2)^2 + (1 - x_1)^2 + 90(x_4x_3^2)^2 + (1 - x_3)^2 \\ +10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{3.15}$$

subject to $-10 \leq xi \leq 10$. The global minima is located at $x^* = f(1, ..., 1)$.

- **Zakhharov Function** [83] (Continuous, Differentiable, Non-Separable, Scalable, Multimodal).

$$f_{Zakharov}(x) = \sum_{i=1}^{n} x_i^2 + (\frac{1}{2}\sum_{i=1}^{n} ix_i)^2 + (\frac{1}{2}\sum_{i=1}^{n} ix_i)^4 \tag{3.16}$$

subject to $-5 \leq xi \leq 10$. The global minima is located at $x^* = f(0, ..., 0), f(x^*) = 0$.

- **Rosenbrock Function** [84] (Continuous, Differentiable, Non-Separable, Scalable, Unimodal).

$$f_{Rosenbrock}(x) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right] \tag{3.17}$$

subject to $-30 \leq xi \leq 30$. The global minima is located at $x^* = f(1, ..., 1), f(x^*) = 0$.

- **Branin Function** [85] (Continuous, Differentiable, Non-Separable, Non-Scalable,

Multimodal).

$$f_{Branin}(x) = f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1 + s \tag{3.18}$$

the recommended values of a, b, c, r, s and t are: a = 1, b = 5.1 / ($4\pi^2$), $c = 5/\pi$, r = 6, s = 10 and $t = 1/(8\pi)$.

with domain $-5 \leq x_1 \leq 10, 0 \leq x_1 \leq 15$. It has three global minima at $x^* f(\{-\pi, 12.275\}, \{\pi, 2.275\}, \{3\pi, 2.425\})$, $f(x^*) = 0.3978873$.

- **Booth Function** [79] (Continuous, Differentiable, Non-separable, Non-Scalable, Unimodal).

$$f_{Booth}(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \tag{3.19}$$

subject to $-10 \leq xi \leq 10$. The global minimum is located at $x^* = f(1, 3)$, $f(x^*) = 0$.

- **Goldstein Price Function** [86] (Continuous, Differentiable, Non-separable, Non-Scalable, Multimodal).

$$\begin{aligned} f_{GoldsteinPrice}(x) = &\left[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right] \\ &\times \left[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x2 + 27x_2^2\right] \end{aligned} \tag{3.20}$$

subject to $-2 \leq xi \leq 2$. The global minimum is located at $x^* = f(0, -1)$, $f(x^*) = 3$.

- **Ackley Function** [85] (Continuous, Differentiable, Non-separable, Scalable, Multimodal).

$$f_{Ackley}(x) = -a\exp\left(\sqrt[-b]{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1) \tag{3.21}$$

recommended variable values are: a = 20, b = 0.2, and c = 2π

subject to $-32.768 \leq xi \leq 32.768$. The global minimum is located at $x^* = (0, ..., 0)$, $f(x^*) = 0$.

The selection is based on two aspects: they are usually part of another set of benchmark functions, and the goal is not to propose the absolute best algorithm but to compare the effect of parameter settings on performance directly. The Jaya algorithm provided the result for 25 populations with a complete fitness evaluation of 500000.

3.3 Experimental Evaluation

Table 3.2 shows the results of the first evaluation using the first method with two different weights. Because this is an initial experiment, the function selection differs slightly from the main experiment. Aside from the two roles, there are no benefits, although disadvantages are conceivable. We cannot think of an effective method to improve Jaya in this way.

Table 3.2: Result obtained by the Jaya algorithm with two different weight.

Function	**Standard Deviation**		
	(102)	***(201)***	***Jaya***
Sphere	0	0	0
SumSquares	0	308.28	0
Beale	0	0	0
Easom	0	0	0
Matyas	0	0	0
Colville	0	0	0
Zakharov	0.00364	0.000033	0
Rosenbrock	0.000014	8888003	0
Branin	0	0	0
Booth	0	0	0
GoldStein-Price	0.000008	0.000007	0
Ackley	0.045268	0.001646	0

The dimension values from the benchmark function are presented in Table 3.3 for equation (3.5) with a weight1 of (0.9, 0.1 0 -0.1 -0.9). According to method 2, the weight alternatives include the second-best tensile force and the second-worst repulsion force. The Beale function, which has five different weights, is one of the unconstrained benchmark

functions shown in Figure 3.2.

Table 3.3: Result obtained by the Jaya algorithm with five different weight.

F	**Dimension**					
	Jaya	***W1***	***W2***	***W3***	***W4***	***W5***
F1	0	0	0	0	0	0
F2	0	0	0	0	0	0
F3	0	0.34	0.0009	0.015	0.0084	0.0008
F4	0	0	0.19	0	0.0002	0
F5	0	0	0	0	0	0
F6	0	0	0.52	0.45	1.13	1.38
F7	0	0	0	0	0	0
F8	0	0	1647.1	7.97	0	0.88
F9	0	0.0003	0.0053	0.0063	0.0008	0.001
F10	0	0	0	0	0	0
F11	0	0.012	0	0.106	0.012	0
F12	0	0.81	0	0	0	0.42

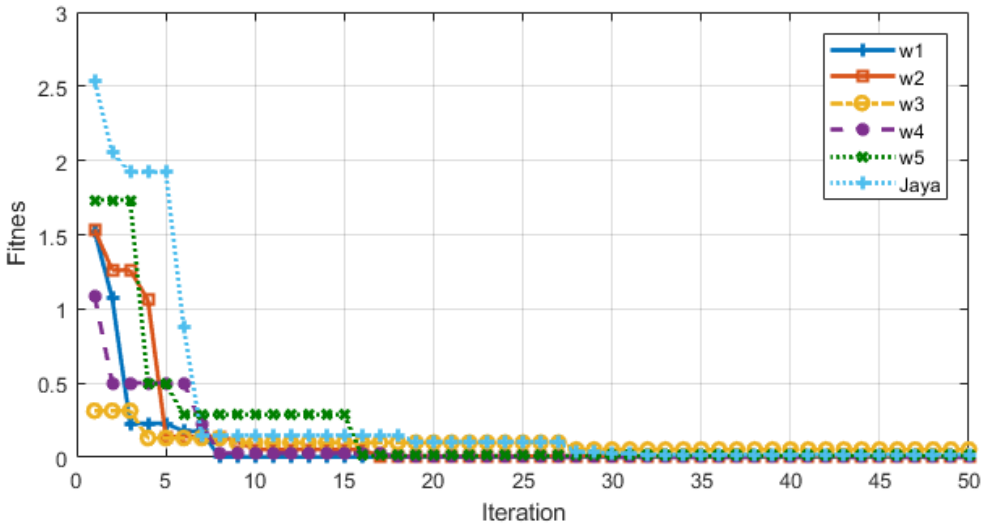


Figure 3.2: Fitness convergence in the case of Beale function with five different ordered weights.

We also evaluated the 95% statistical significance for the data presented in Table 3.3. The initial weight (0,9 0,1 0 -0,1 -0,9), the p-value is 0.1875 indicates that Jaya with standard parameter settings is not significantly better (only relatively better), whereas for other weights ((0.7 0.3 0 -0.3 -0.7), (0.5 0.5 0 -0.5 -0.5), (0.3 0.5 0 -0.5 -0.3), (0.8 0 0 0 -0.8) the p-value of 0.0625 indicates Jaya with the basic parameter settings significantly better than Jaya with the additional parameter settings since the p-value normally is 0.05., as shown in Table 3.4.

Table 3.4: Test on statistical significance for Beale function using Wilcoxon Signed-Rank test.

Weight	V	P-Value
(0.9 0.1 0 -0.1 -0.9)	2	0.1875
(0.7 0.3 0 -0.3 -0.7)	0	0.0625
(0.5 0.5 0 -0.5 -0.5)	0	0.0625
(0.3 0.5 0 -0.5 -0.3)	0	0.0625
(0.8 0 0 0 -0.8)	0	0.0625

This study's experimental results demonstrate that Jaya still performs best without weighting parameters for most functions, as present in the paper [87]; the Jaya method analyzes different optimum power flow (OPF) problems. The authors utilize the basic Jaya algorithm in this study, with no additional controller parameters. The results show that the Jaya algorithm can provide an optimal solution with expeditious convergence, and when compared to other stochastic algorithms, the Jaya algorithm leads in terms of solution optimality and feasibility, demonstrating its efficiency and potential.

3.4 Discussion

After experimenting with different weight values, the study perspective on Jaya becomes more convincing regarding the parameter controller. Furthermore, the fitness convergence plots are compared and examined extensively to understand better how the weights impact the search in the Jaya process. This effect allows us to predict how fast the stagnation will

occur (being trapped in the local minima or unable to locate a better solution because the search is too dispersed).

As previously stated in this study, several sections of Jaya's formula were modified to see the data differently. The modification was to attempt the Booth function formula without absolute values (abs), which resulted in best = 0, worst = 0.000001, mean = 0, and standard deviation = 0. We can achieve a good result even while utilizing the sinus as a coordinate function. We also test the Beale function, as given in Table 3.5, and acquire satisfying results.

Table 3.5: Modified Jaya formula used Beale function.

Function	Best	Worst	Mean	SD
Square	0	0.000003	0.000001	0.000001
Log	0	0.000006	0.000002	0.000002
Sin	0	0	0	0

SD: Standard Deviation.

We may conclude from this experimental research that Jaya's basic algorithm still works effectively without additional parameters. As one of the most well-known optimization algorithms at the moment, there are still certain elements that need to be researched, such as the influence of the coordinate function, because it appears to be capable of improving performance in some cases. Aside from that, the Jaya algorithm still requires comprehending how this seemingly simple and good algorithm performs.

The purpose of this research is to have a clear insight of Jaya in terms of stochastic/anti-gradient-based search. The average values of these alternative coordinate functions are connected to individual coordinate values, with random variations around these values. The search in the general gradient-derivative learning method will be for a change in the value of the most potent objective function. We can see the composition of this direction with the opposite direction of the most significant loss here (called anti-gradient now). Therefore, in conclusion, more research is required to determine how much this improves the vision and provides a means to comprehend and plan Jaya's application in practice (as well as the design of other algorithms or modifications of known algorithms in a similar sense).

This study also teaches us about the importance of parameter setting in MHS to determine whether the algorithm does not become too difficult to use because we have to find and set many parameters.

3.5 Summary

In this chapter, we propose modifying the parameters of the basic Jaya algorithm to see whether Jaya is parameter-free. Related to the concept of Jaya, where the solution obtained for a given problem moves towards the best solution and away from the worst solution, we first want to investigate the efficient but straightforward Jaya formula, which is based on the best and worst solutions simultaneously, more specifically, which of both is the most effective in the optimization problem.

Then we introduce the weights, which will have an equal number of weights on both the best and worst sides. Finally, we examine the second best and second worst impacts, which gives us a decent idea of how the algorithm operates. The performance of the algorithm is evaluated using 12 unconstrained benchmark functions.

The experimental results confirm that Jaya's algorithm still performs better without weighting parameters for some functions. After altering the weight values, the results of this experiment also reveal Jaya's perspective more clearly.

Chapter 4

Embedded Metaheuristic Search Algorithms for Blockchain Proof-of-Work

Following the description of the metaheuristic search (MHS) algorithm and parametric studies in chapters 2 and 3, we suggest implementing one of the MHS in the second research approach, which is to embed it in the Blockchain Proof of Work in this section. The description covers an introduction to Blockchain and Proof of Work, the problem of excess energy, a solution suggestion, and lastly, the implementation of the MHS method to the problem.

4.1 Introduction

The concept of Blockchain, particularly as a ledger for bitcoin transactions, has resulted in massive amounts of electrical energy being wasted performing Proof of Work (cryptographic puzzles) tasks. Considering these difficulties, we propose an alternative approach for this energy to be spent, at least for a useful purpose, solving a real-world optimization problem.

This chapter focuses on the Blockchain field by proposing an alternative concept of using an iterative optimization algorithm to provide the Proof of Work (PoW) required to

extend the Blockchain with new blocks and implement it using the Traveling Salesperson Problem as a problem model. The basic idea is to lower the tour cost for the best tour identified for block n, which is then expanded by adding one more city as a prerequisite for including a new block in the Blockchain. This solution enables the design of restricted Blockchains while also solving the underlying combinatorial optimization problems. This problem necessitates using optimization approaches that are more efficient on their own and can compete in the real world. The MHS algorithm is an exciting class of optimization algorithms that may be used in the proposed method. The increasing complexity of the issues addressed by binary PSO is also proven by numerical tests, which are the primary need for the entire concept to work in practice.

This chapter present a system that uses optimization problems as a PoW instead of the more common cryptographic puzzles currently in use. Our method is built on the fundamental idea that we want to solve the optimization problem iteratively, with optimization algorithms operating as references. The Traveling Salesperson Problem (TSP) is the optimization problem that we employ in this study. TSP is a common NP-hard problem in determining the best path that different methods can solve. We aim to discover a tour of all nodes in a weighted graph that minimizes the overall weight in TSP.

The chapter is structured as follows: Section 4.2 is an overview of the background of this research, Section 4.3 then introduces the concept of embedding optimization tasks into Blockchain processing. Section 4.4 then discusses concerns relating to the practical consequences of the suggested method and presents the results of various feasibility studies, and the chapter is summarized in Section 4.6.

4.2 Blockchain as Public Ledger of Bitcoin Transactions

4.2.1 Blockchain

The Blockchain idea was implemented in the cryptography area before being utilized in cryptocurrencies. Afterward, Blockchain was developed as a distributed ledger system to support the Bitcoin currency. This technology enables users to make direct transactions (peer-to-peer) without requiring the involvement of a trusted third party [88]. Blockchain

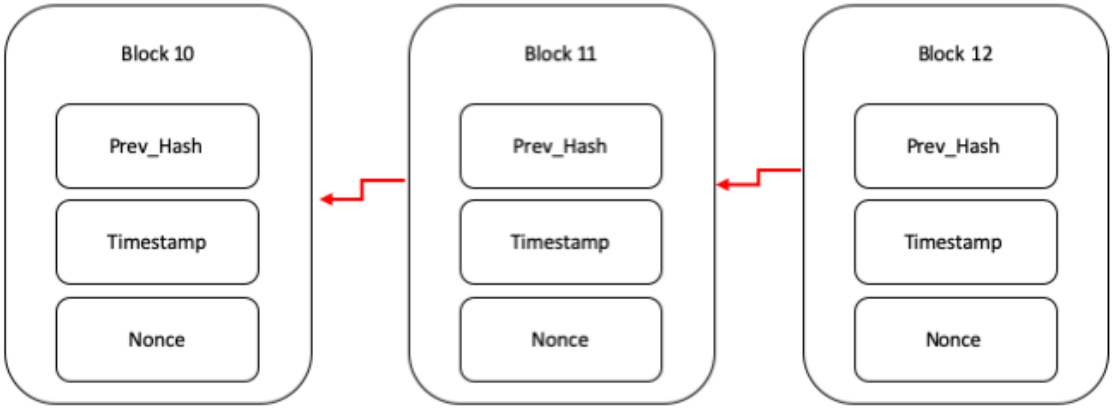


Figure 4.1: A Blockchain example, consisting of a continuous block sequence.

security is generally from arranging a cryptographic puzzle that adds (mining) new blocks to the Blockchain. However, this consensus process was having significant issues with squandering too much power energy since miners had to perform many computer calculations [89].

As a distributed ledger, Blockchain is essentially a distributed system that can be centralized or decentralized. The distributed ledger here means spread across throughout the network, with each partner in their network having a copy of the complete ledger. Nodes in this distributed system can be configured as individual players, and messages from and to each other can be sent and received. In addition to memory and processing nodes, it might be destroyed or harmful due to dishonest user behavior.

The Blockchain's original shape is a hash tree, also known as the Merkle tree, developed in 1979 by Ralph Merkle [90]. This data format can be used for data verification and management across computer systems. Data validation is critical in peer-to-peer networks to guarantee that nothing has changed during the transmission process and prevent the delivery of false data. This procedure is essentially used to protect and demonstrate the integrity of the shared data.

Blockchain works as a distributed database to maintain a continuously expanding list of blocks. A timestamp and a link to the preceding block are consist of each block [91]. Figure 4.1 is an example of a Blockchain structure in which each block is identified by a cryptographic hash and relates to the preceding block's hash. This procedure connects the

blocks that constitute blockchain.

In 2009, bitcoin developed as an electronic cash system (e-cash), and the word cryptocurrency first surfaced to answer the challenge of distributed consensus in a network without trust. The PoW mechanism with public-key cryptography is used to offer a safe, regulated, and decentralized way for generating digital currency, with the core innovative concept being a blocklist generated from transcripts and cryptographically protected by the PoW process. Miners must compete to discover numbers more minor than the objective of network difficulties that are PoW requirements and add new blocks to the Blockchain. The difficulty in determining the correct value is often referred to as a cryptographic puzzle [92]. In the next section we will explain more about PoW.

4.2.2 Proof of Work

PoW is one of the standard method to reaching consensus on Blockchain by solving complex cryptographic puzzles but are easy to verify. The difficulty of PoW may be modified to reflect a Blockchain's long-term growth and increased technological capabilities. Currently, the block rate for Bitcoin, as an example of a real-world Blockchain application, is about one block every 10 minutes [93].

Several consensus protocols are shown in Figure 4.2 have been proposed, some of which are well-known: PBFT [94], Proof Of Stake [95], and Proof of Elapsed Time [96]. However, most of the existing Blockchain still uses the computationally expensive PoW consensus mechanism and presently represents over 90% of its entire warehouse capitalization of existing digital money [97]. The security of PoW is based on the premise that no entity should have more than 50% of the processing power since such an entity may effectively control the system by maintaining the longest chain.

In PoW, each network node computes the hash value for the block header. The block's header contains a nonce, which miners regularly modify to have different hash values. The block is broadcast to all other nodes when a node reaches the target. Following that, all other nodes jointly validate the hash accuracy. Lastly, other miners will add the verified block to their Blockchain [98].

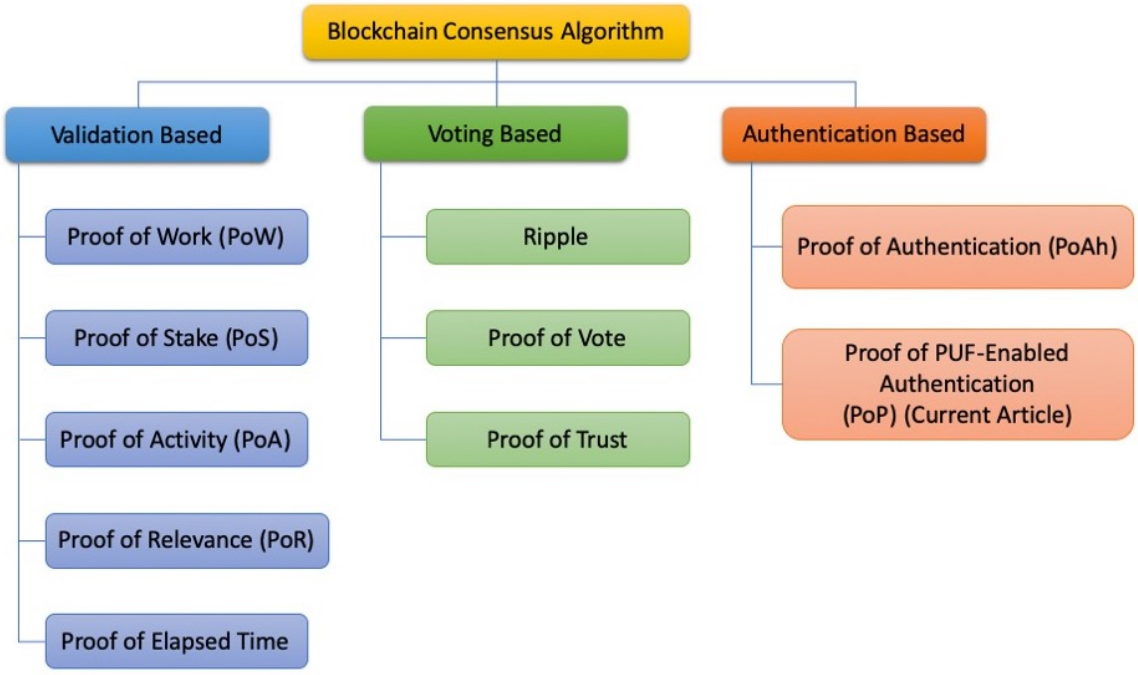


Figure 4.2: Various consensus algorithm in Blockchain.

4.3 The Concept of Embedding Optimization Tasks Into Blockchain

4.3.1 Proposed Methodology

In this part, we explain in detail the proposed concept. The suggested scheme is shown in Figure 4.3, which serves as an optimization problem for PoW rather than today's more common cryptographic puzzles. Each block generally consists of the TSP memory hash of the previous block, as detailed in the steps below. The suggested SOLVER scheme encodes a unique solution for the specified number of cites in each stage/block of the Blockchain process, as shown in the image using various appropriate optimization algorithms. The following steps summarize the detailed procedure of the proposed scheme:

- Initialization: The Genesis Block (first block in the new Blockchain) is set up with the first Blockchain data record (derived from the specific application of the Blockchain), TSP memory that includes a complex TSP instance given by the locations of cities,

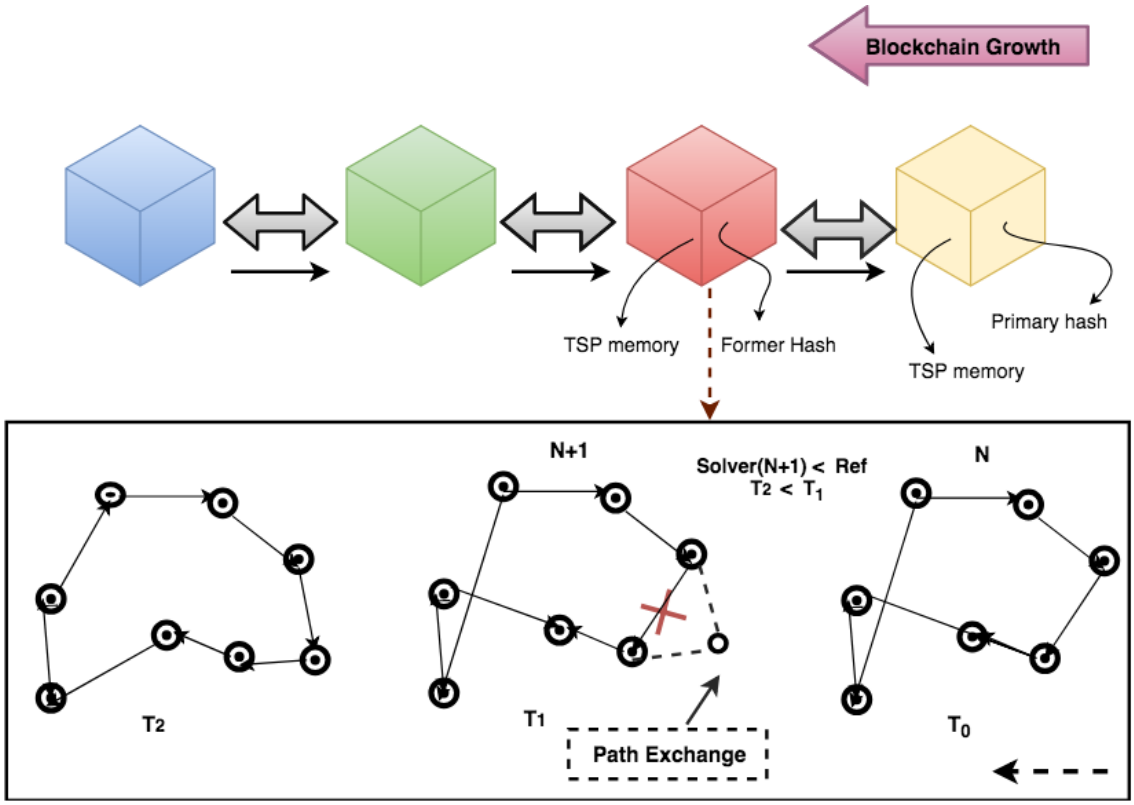


Figure 4.3: General procedure of the proposed scheme.

and the hash value of all data. Then, a sufficient number of cities from TSP memory is selected, and SOLVER finds a good tour between those cities. ”Good” and ”sufficient” here mean to have solved a challenging task right from the beginning. The path p_1 of the found tour is stored.

- With a Blockchain of n blocks and a request to add the following block (which already includes a data record and a previous block's hash), block n transmits the stored tour between n cities to block $n + 1$.

- The new block randomly selects one more city from the TSP memory and extends the path p_n by this new city. The new city can be linked between the first and n-th city of the former path (but other schemes might be applied). The tour cost T_1 of the extended path is calculated and used as a reference value in the following.

- SOLVER is now given the task of finding a tour among the $n+1$ cities with a total cost of $T_2 < T_1$. This task was supposed to be a challenging task by proper configuration.
- The new path found is stored in block $n + 1$.
- If the algorithm gets stuck (for example, the expanded tour of path p_n is accidentally already the optimal one), the process must be restarted by selecting a different city.

4.3.2 SOLVER Classes

The suggested Blockchain idea requires appropriate optimization algorithms. There exist exact methods for solving all types of combinatorial optimization problems (COPs), including the TSP. The challenge here is that a Blockchain designed to solve optimization problems like PoW should be scaled such that those algorithms are no longer appropriate. In other aspects, exact SOLVERs would be detrimental to the approach, and the optimization problems must be defined so that the work required to discover an accurate solution is exceedingly high. However, this can be readily accomplished by increasing the number of cities. In this section, we discuss various considerations for choices and design alternatives.

This argument requires the use of metaheuristic algorithms. Operating under the general soft computing paradigm that higher effort brings a higher precision, such algorithms can approach optimal solutions to a degree defined by their configuration, most at all population sizes and several generations. However, as a brief literature search indicates, most algorithms are designed to solve persistent unconstrained optimization problems. In general, a transition to the discrete domain of COPs requires certain modifications in the internal structure and the method for representing a problem. Notably, the operator design to handle TSPs using metaheuristic method resulted in various somewhat complicated approaches [99] with limited effectiveness.

In order to create appropriate metaheuristics, at the very least, a starting point is required. We focus on two critical issues: encoding of solutions and fitness evaluation.

There are both general and basic encoding techniques available. Assume the solution (particle, gene, ...) given as a vector x of n elements x_i ($i = 1, \ldots, n$), where n is the number of cities of a TSP. So, we must map a real-valued vector to a permutation of the numbers from 1 to n that represents a tour.

- **Asymmetric encoding**: The x_i is from the range $[0, n-1]$ and in the first step, we round all values, thus having a set of n integers. Then, $x_1 + 1$ is the index of the first city in the tour. Next, after removing this city from the index set, $x_2 \, mod \, (n-1) + 1$ is the index of the 2nd city in the tour among the remaining indices, and so on, $x_k \, mod \, (n-k+1) + 1$ the index of the k-th city among the remaining ones in the tour. For example, having three cities, and after rounding the solution $x = (1, 0, 2)$, it refers to the path $p = (2, 1, 3)$. The problem here can be that the impact of each solution component differs. For example, the value of the last number does not play a role as the selection is always the last city remaining, while the first city is directly selected (therefore, we call it asymmetric encoding).

- **Order encoding**: In this method, the influence of each component can be increased. The index order corresponds with the ordering of the solution components and using the original position order in the solution vector in case of ties. For example, a vector (there is no need to round or restrict the range) $x = (0.25, 0.05, 0.13, 0.76, 0.05)$, using minimum ordering, would represent a path $p = (2, 5, 3, 1, 4)$ (x_2 and x_5 are equal, so they are kept in the original order). Here, all components have the same influence on the absolute path, but their magnitude does not matter on the minus side.

- There might be alternatively the interest to convert the real-valued solution vector first into a bitstring. The reason behind it can be that over the years, much experience has been accumulated to handle COPs by Standard Genetic Algorithms. This conversion is usually achieved by thresholding (e.g., for the PSO in [100]). The metaheuristic algorithm maintains internal processing based on real-valued vectors under one restriction: the components are from the range $[0, 1]$. Then each time it comes to a fitness calculation, the components are thresholded by 0.5, and the corr. component of the bitstring becomes 0 if it is smaller than 0.5 and 1 otherwise. In a variation of this method, removing the obligation to use 0.5 as a threshold, we are currently investigating a generic algorithm design where a co-population of threshold vectors is added to the algorithm. Then, in each iteration step, the thresholds are evaluated parallel to the individual solutions population. The average fitness of all individuals determines the fitness of a single threshold if using that threshold for

converting the individual real-valued vector into a bitstring (to make it clear, if t_i is the i-th component of the thresholder, than x_i is replaced by 0 if $x_i < t_i$ and by 1 otherwise). Then, evolutionary operators are applied to the threshold population in a standard way. Next, the best threshold is used to convert all individuals into bitstrings to get the individuals' fitness values needed for the original metaheuristic. After a termination criterion is met, the thresholds can be thresholded themselves, simply by 0.5, in order to get another bitstring solution in a reversed fashion: small components indicate that the corresponding bit is most likely set to 1, while components close to 1 indicate a most likely bit value of 0. This is a rough overview of the threshold concept results with that procedure a subsequent study presented in a forthcoming publication.

In summary, using such methods, any metaheuristic can be made suitable to handle the corr. Blockchain optimization problem in an approximating manner. After all, it can trigger the suggested technique to work as a benchmarking tool for a direct comparison, leading to superior algorithms. It is opposed to the present technique of solving cryptographic puzzles, in which, aside from identifying systemic flaws in hashing algorithms, there seems not much contribution to any cryptographic methodology. Simply said, the authors hypothesize that using the suggested concept in real-world practice would drive the development of powerful new algorithms.

4.4 Experimental Evaluation

PSO-based optimization is used to get the best solution for each stage of the proposed Blockchain system. The 10000 route instance used to solve the TSP problem for 30 cities is demonstrated in Figure 4.4. All values in the graph are not highlighted to provide a clear illustration. However, the graphic indicates that an optimum objective with a smaller total distance is selected after multiple iterations.

Furthermore, Figure 4.5 illustrates the optimal route result (optimum distance) determined for each city. This experiment involves 10000 iterations, 10 particle counts, a maximum velocity of 4, and an objective number of 86.63. At each stage of the Blockchain

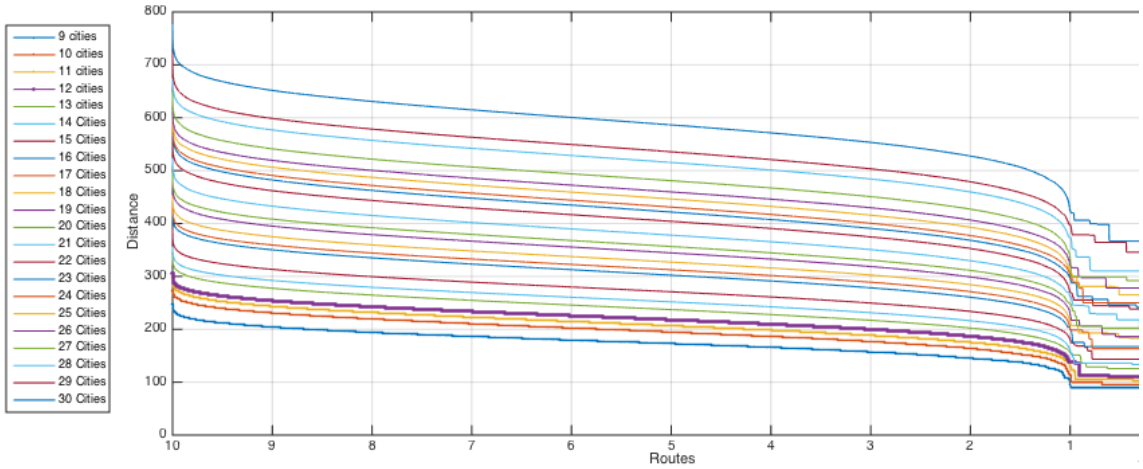


Figure 4.4: PSO optimized routes for TSP problem with 10000 iterations.

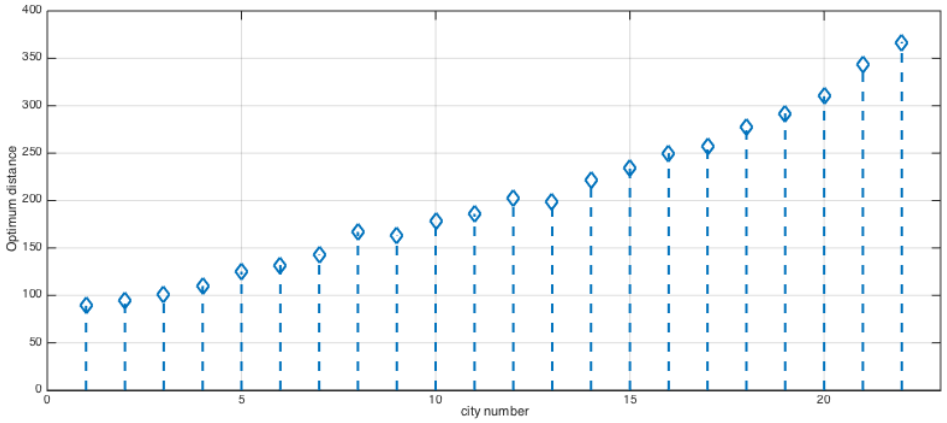


Figure 4.5: The optimum route result (optimum distance) selected for each number of the city.

process, the optimal route is determined and encoded as PoW. One of the routes chosen for Blockchain number 22 is : 0, 19, 7, 18, 12, 5,13,4,11,2,15,1,20,3,17,10,8,6,9,14,16, with a distance of 199.034. The encoding might differ from block to block, but as mentioned in step 4, the core idea requires a binary conversion of each city route.

As stated in the previous paragraph, Blockchain aimed at solving optimization problems as PoW should be scaled to determine the result as tricky as feasible. For each specific part of the Blockchain scheme, the whole concept for adding security features for the Solver

algorithm is based upon finding the unique solution for each step of the TSP problem. The experimental results demonstrated that utilizing the concept suggested can lead to new, robust systems in the real-world scenario.

However, as previously stated, any metaheuristic may be adapted to deal with combinatorial optimization problems, including TSP. As shown in Figure 4.6, another metaheuristic optimization method known as Quantum Behave PSO (QPSO) is used to analyze the best solution for the TSP problem [101]. As previously stated, the QPSO belongs to the BBPSO family, which evaluates each particle location using a double exponential distribution.

This study offers a mechanism for resolving complicated cryptography challenges that can affect how current method work. Figure 4.6 demonstrates several optimized results (best cost) in terms of the total number of iterations necessary by SOLVER in each block of the proposed Blockchain to reach the optimal value (best route). We selected a limited number of populations (cities) with the exact coordinates as prior trials for demonstration reasons.

The behavior of each particle, however, is affected by how the value of α modifies. In conclusion, the following parameters have been chosen for this particular experiment: Min $\alpha = 0.08$ and Max $\alpha = 0.9$ for the following city locations:

$X = [30, 40, 40, 29, 19, 9, 9, 20, 10, 5, 30, 23, 8, 12, 4,$
$42, 25, 31, 8, 29, 39, 41, 38, 27, 28, 15, 14, 7, 37, 3]$

$Y = [5, 10, 20, 25, 25, 19, 9, 5, 5, 5, 12, 20, 3, 20, 15,$
$6, 11, 18, 4, 9, 12, 21, 17, 16, 24, 13, 26, 32, 23, 41]$

4.5 Discussion

The experimental results demonstrate that as the number of cities increases (the number of cities in each block increases), so does the number of iterations, and the optimized cost decreases. However, depending on the number of cities, the objective can be met with fewer iterations. For this experiment, we set the total number of iterations for each city is to 400. In figure 4.6 the total number of particles is set to 100, Minimum Alpha: 0.9, Maximum Alpha: 0.08, total iteration: 400, $alpha = (maxalpha - minalpha) \times (MAXITER - t)/MAXITER + minalpha$. However, this number can vary depending on how many cities

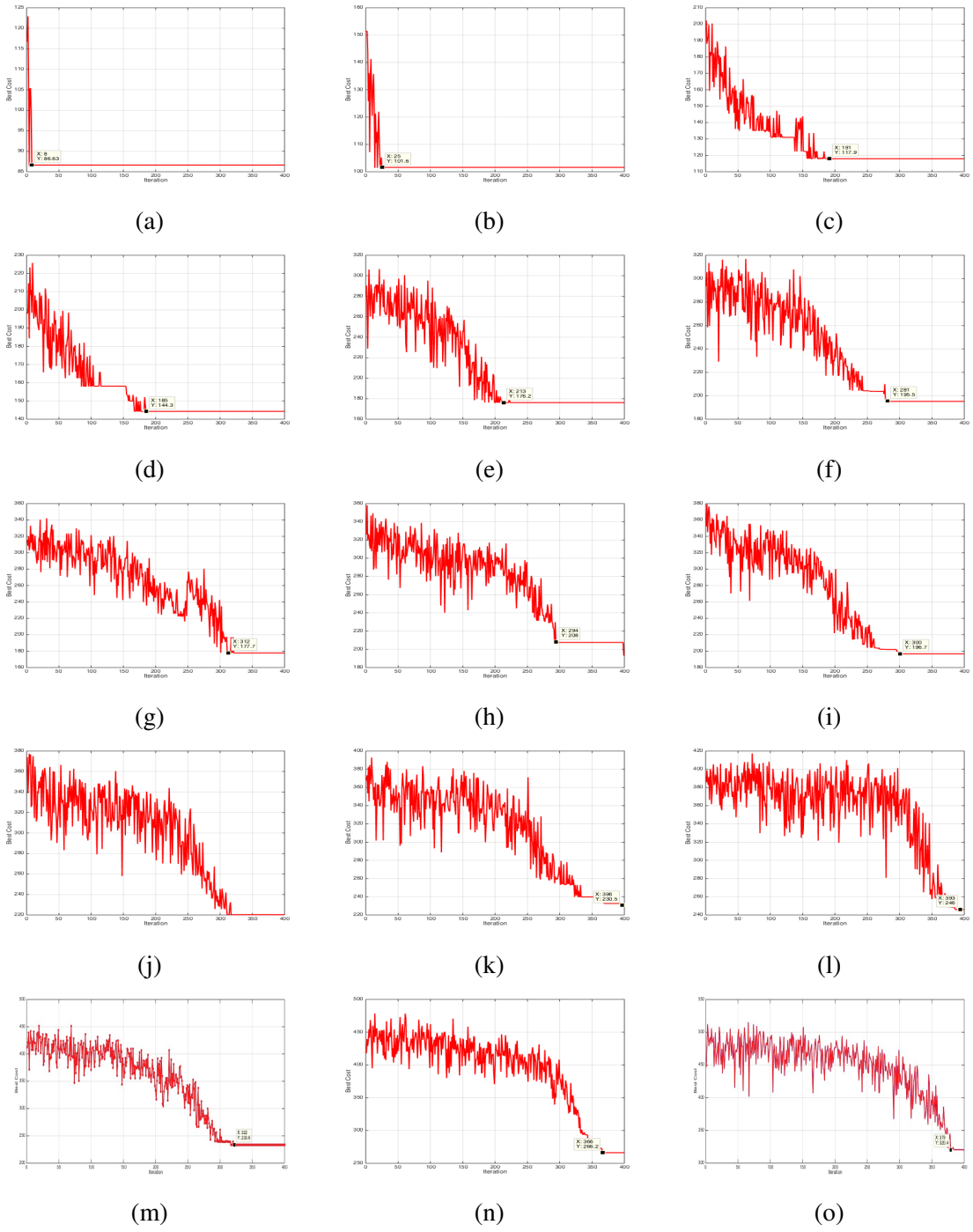


Figure 4.6: Simulation result of QPSO for TSP problem for different number of cities.

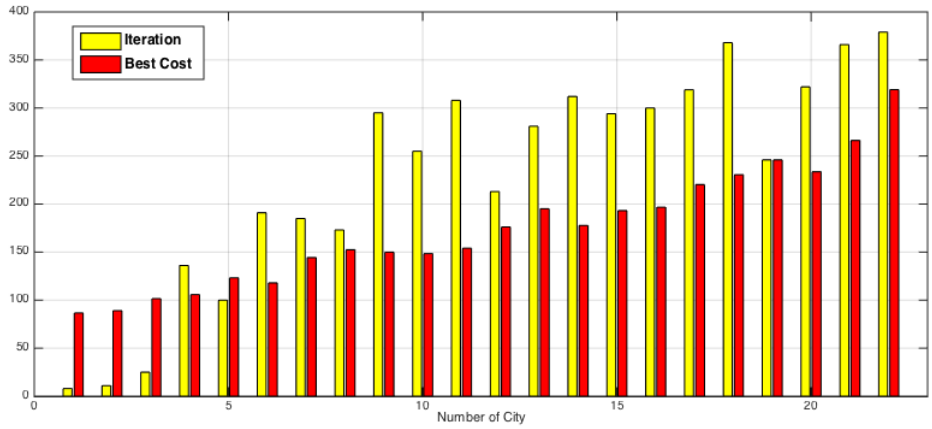


Figure 4.7: The frequency of the iteration number versus optimized cost value for TSP problem.

are involved in the specific TSP problem. From figure 4.6a to 4.6o it show the changes in the iteration to optimize the performance of the different number of cities in each block show a significant difference in the number of iterations and the reduction of optimized costs.

Figure 4.7 represents the frequency iteration number versus optimized cost value (using the QPSO optimization algorithm) for the TSP problem. The results show the differences in iteration and the best value in each Blockchain block. Figure 4.7 also indicates that the number of iterations required to reach the next level, thereby adding a new block, rises in a relatively linear way. This solution also allows the Blockchain concept to be scaled so that it is neither complex nor straightforward to solve. In conclusion, several MHS optimization techniques may be used in the proposed method to create a more straightforward and more secure design of restricted Blockchains.

In a recently published journal, author López [102] shows how MHS address Cryptographic Boolean Functions (CBFs) problems. Developing (CBFs) with high nonlinearity is a challenging task, and several heuristic methods have been proposed to solve it, but the results obtained so far have not been as effective as those obtained using algebraic techniques. The author López presents a set of trajectory-based and population-based MHS

approaches for producing CBF with high nonlinearity. Based on this problem and our findings, we assume that MHS algorithms can address complicated cryptographic problems by integrating diversity-based MSH.

According to the findings of this study, the MHS search algorithm has shown to be effective in implementing PoW on the Blockchain. We proposed TSP as a problem model instead of solving usual cryptographic puzzles in PoW. When we implement this concept to the Blockchain in the future, every block will consist of a TSP memory hash and other continuous data. Since TSP is used to generate the block, miners can save time and energy to solve hashes on the Blockchain.

4.6 Summary

Blockchain technologies have recently caught the interest of several researchers from across the world. This technology enables direct (peer-to-peer) transactions to occur without the involvement of trusted third parties. However, the Blockchain's security may be enhanced based on a cryptographic-based method known as PoW. This chapter presented a method for providing the PoW required to add a new block to expand the Blockchain using iterative optimization algorithms. TSP is used as the optimization problem in this study. Implementing the appropriate MHS approach in the suggested SOLVER class can develop a secure and influential algorithm.

We may conclude from the results of the study that SOLVER (the suggested approach) can transform the way present systems address complicated cryptographic problems. This approach also allows for the development of algorithms based on the Blockchain idea so that it is not too easy and secure but also not too complex to solve.

Chapter 5

Idle-Metaheuristic for Flower Pollination Simulation

Later in this section, we present a second research approach that includes suggested metaheuristics search (MHS) algorithms that have been adjusted to be relevant to this second field. The description covers the real-world problem of cocoa flower pollination, the proposed Idle-Metaheuristic, and the implementation of the selected method to the flower pollination simulation.

5.1 Introduction

Many optimization techniques have been created, inspired by nature, and adapted to answer various optimization problems since nature is a good source of inspiration for optimization method. In this chapter, we presented a MHS to be used in a new field inspired by nature, specifically pollination optimization in cocoa plants.

Pollination is an essential biological cycle that the practically most plant requires to produce seeds and fruit. One of them is the cacao plant, whose fruit production success is almost entirely dependent on pollination [103], and we utilize the cacao plant as a model of the challenge to be solved through optimization in this study. We can find the cacao plant (*Theobroma Cacao*) almost anywhere, but the most prevalent is in tropical places, including West Africa, Indonesia, Central and South America, and Hawaii. The term cacao can be

interchangeably used with the term cocoa, as we will also do here in the following. Because cocoa flower pollinators favor humid conditions, this geographical issue is a challenge for chocolate manufacturing. Another issue concerning cacao plants is that the flowers are so small and distinct from other flowers that only a few flies genera, most notably a tiny midge called *Forcipomyia Inornatipennis* (FP), can pollinate them. Because the cacao flower is tiny and nearly odorless, it does not draw the interest of many insects, mainly traditional pollinators like bees.

The purpose of FP is to explore cacao flowers more regularly than other insects to collect nectar for feeding and egg maturation. The pollination process in cacao flowers may be illustrated as follows: whenever the FP collects nectar from the cacao flower, the FP unintentionally contacts the bunch's head, leading pollen to be expelled and adhere to the FP. The pollen adhering to the FP acquired from one flower then meets the pollen contained on another cacao flower visited by the FP, so this process is causing accidental pollination.

Cocoa plants are self-incompatible tree species, which means they fertilize flowers on other trees, including the same type known as cross-compatible. Since self-pollination is not optimal for cacao plants, the only acceptable way to produce adequate fertilization is through cross-pollination [104]. Due to this issue, the purpose of this study is to analyze this process by simulation pollination on cocoa plants utilizing three pollinator collaboration method:

1. Based on Swarm Intelligence Algorithms;
2. Based on Individual Random Search;
3. Based on Multi-Agent Systems Differential Search Method.

The study focused on theoretical and empirical research into method required to analyze stochastic optimization algorithms and performance evaluation based on several criteria. The following briefly describes the process of pollination of FP in cocoa trees based on Figure 5.1:

- The FP can employ various search strategies to guarantee that the FP collects enough nectar.

- However, if the tree has been pollinated, the flowers will no longer be exposed, and the tree will no longer produce nectar.

- As a result, FP needs to search out different unpollinated trees to collect more nectar.

This procedure is then performed in a 3D virtual environment to illustrate and evaluate the efficiency of various cocoa pollination method. The following is how this chapter is arranged. The real-world facts about cocoa pollination are presented in Section 5.2. The resources and methods used in this study are detailed in Section 5.3. Section 5.4 presents the experimental results, and last is the discussion, summarizes of this chapter in Section 5.5 and 5.6.

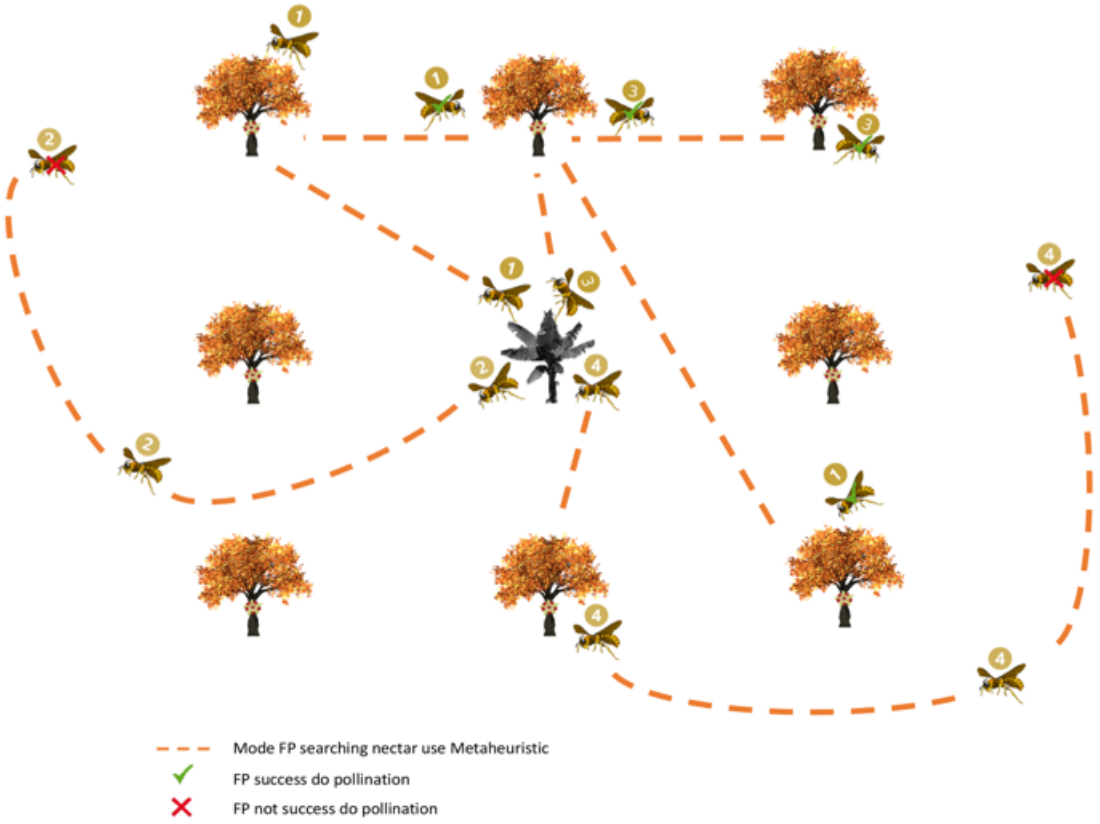


Figure 5.1: Process flow of *Forcipomyia* pollination.

5.2 Optimization Problem in Real-World Pollination of Cocoa Flower

In comparison to flower pollination in general, cocoa flower pollination employs a different pollination process. Since the flowers on these cacao trees are different from other plants because they are tiny (average diameter of around 3 cm), allowing only small-bodied insects to pollinate them (most excluding traditional pollinators such as bees). Cacao flowers have a distinctive hooded petal that conceals the stamens, making self-pollination and cross-pollination difficult. However, there is abundant evidence from multiple sources that natural cross-pollination occurs in significant quantities [105]. According to the observations of Jones (1912) in Dominica, it is apparent that tiny insects, whether ants, aphids, thrips, or a combination of the three, are the significant agents engaged in cocoa flower pollination [106].

Swarms of *Forcipomyia Inornatipennis* can be divided into two types: (1) normal swarms and (2) mating swarms. [107]:

1. **Normal swarm**: The usual FP flying cycle lasts 12 hours. The highest amount of insects are affected by swarming activity between 5 a.m. and 8 a.m., after which it swiftly drops to its lowest level from around midday to approximately 2 p.m. when it begins to climb to a second peak between 5 a.m. and 6:30 a.m. However, insect activity varies according to the amount of light available. The dawn swarm will not start until after 9 a.m. if the sky is cloudy. A swarm of 4–80 individuals of both sexes can fly in either direction within a 100 cm radius, can be seen from 30–180 cm above the ground. The bigger the swarm, the more midges leave; consequently, the remaining population is exactly proportional to the number of individuals involved in the hive.

2. **Mating swarm**: At dusk, almost 60% of mating swarm activities take place. Depending on the day, the swarm can have 2 to 30 individuals, and both hives contain both sexes. Males aggressively search for females in flight, and male mates in a swarm often spend 15 minutes with three or four females. This behavior is because the number of females on flights is always lower than the number of males. If the

(a) (b) (c)

Figure 5.2: Three variants of the cocoa flowers. **(a)** Converging. **(b)** Parallel. **(c)** Splay.

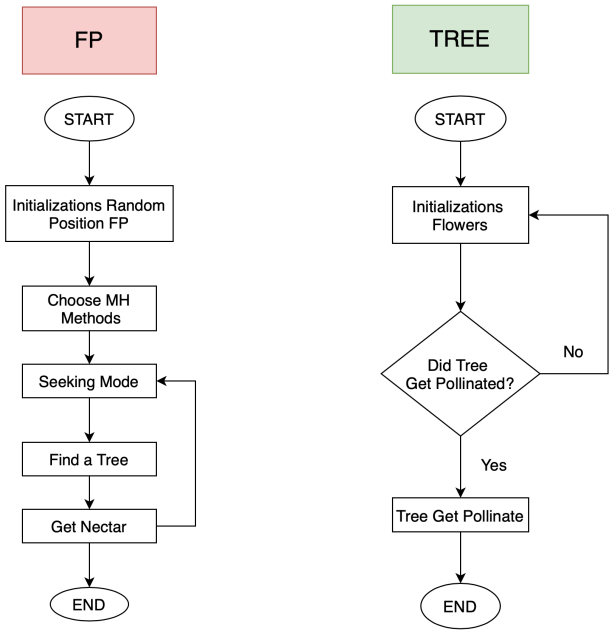
swarm does not comprise more than four or five pairs, the number of remaining midges will be affected by the size of the mating swarm for as long as it persists.

Cocoa flowers are not like other flowers in terms of shape, causing them unappealing to specific pollinators. As seen in Figure 5.2, cocoa flowers feature several staminodes. These staminodes resemble stamens that are the male portion of the flower, but they do not produce pollen and are hence sterile. Only small insects can get to the pollen-producing anthers, which are covered under a hood. Frimpong-Anin's [108] research stated that the variations of convergent and parallel staminode flowers preferred staminode-style flowers for pollination.

Pollination in flowers is an enthralling natural phenomenon that has attracted several academics who have investigated it. Flower pollination's objective is to preserve the most appropriate and ideal plant reproduction in terms of both amount and quality. As mentioned earlier, all of the flowers' pollination characteristics and processes cooperate to maintain that flowering plants reproduce efficiently. As a response to this, the author Xin-She Yang [109] proposed a novel algorithm based on flower pollination.

Aside from that, from author Diane R Campbell [110] created a pollination simulation model, and the findings indicated that pollinator and pollen carrier movement patterns impact seed production. Also Alan Dorin and the team [111] was motivated by the connection between pollinating insects and flowering plants and presented an agent-based simulation to evaluate the possible effect of heterospecific pollen transmission by insects on two flowering plant species in an environment by including a shared central area and specific-species refugia.

The previous explanation concludes that the FP system for pollination of cacao flowers



(a) (b)

Figure 5.3: Flowchart of the pollination model used here: **(a)** FP. **(b)** Tree.

differs from the features of flowers in general. Therefore, three approaches are presented and then analyzed against the FP search method to perceive how they perform. Several researchers have observed that every living thing in our world has a unique selection behavior [112]. Based on this conclusion, a MHS algorithms is provided to support FP in exploration.

We utilized a simulation to execute the FP's pollination methodology based on the principles and concepts provided in [107]. We build a setting that resembles the actual scenario, such as a cocoa tree with small flowers and FP insects, and Figure 5.3 represents the flowchart of our proposed method. The FP and tree have a complicated relationship because the tree cannot self-pollinate, and the FP only needs nectar from the cacao flower, as previously stated. As a consequence, the flowcharts for FP and the tree both follow separate flowcharts at the same time. Figure 5.3a demonstrates how the FP searches for nectar by starting at random positions before deciding on the method to be employed, whereas Figure 5.3b demonstrates how the FP can accidentally pollinate a tree by carrying

pollen attached to its body to a separate tree that has not been pollinated. The pollinating circumstances in this Figure illustrate that pollination occur when an is FP gets closer to a non-pollinated tree, the nectar level of the FP increases, and the FP brings pollen from that tree. The flower is pollinated if the FP already delivers pollen from another tree.

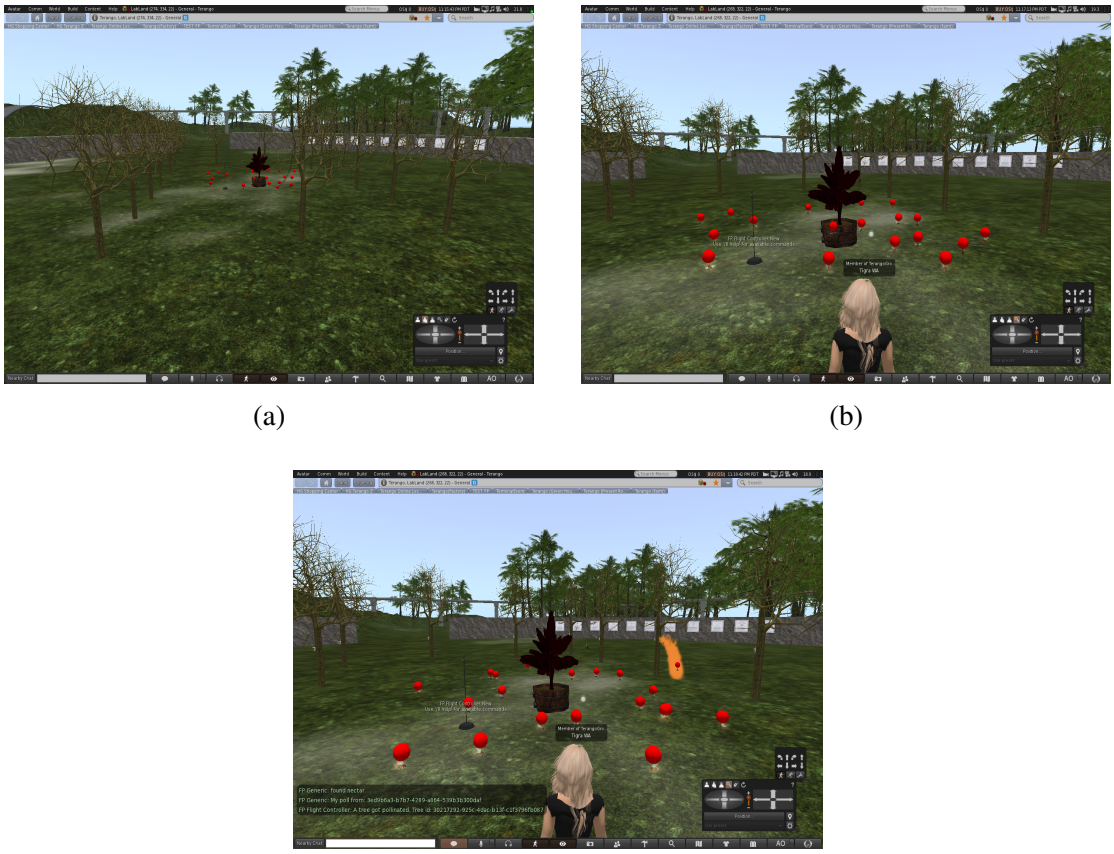
5.3 Idle-Metaheuristic embedded to Flower Pollination Simulation

This chapter explain the resources and methods that we proposed in this study in detail. The simulation was performed on a cloud-hosted instance of the OpenSimulator server (version 0.9.1). This OpenSimulator provides an appropriate research platform by supporting multiple frameworks such as server-client architecture, grid architecture, avatar-based control, concurrency, and scripting support. It became feasible to build an experimental framework for conducting simulations that can be evaluated, analyzed, and updated through multi-institutional collaboration inside the so-called hypergrid connecting the various server simulations globally [113].

For this study, all the experiments were conducted using a Dual-Core Intel Core i5 MacBook @ 3.1 GHz with 8 GB 2133 MHz LPDDR3 of RAM running the viewer (client) software Firestorm-Release X64. Figure 5.4 shows that the OpenSimulator server was operating on the Metropolis Metaversum Grid, hosted by Hypergrid Virtual Solution UG.

5.3.1 Experimental Environment

The experiment goal is to mimic the behavior of FP in the real world and then embed it in a simulation by using a MHS algorithms method for exploration. The use of multiple metaheuristic searches in this research study enables a nearly comparable real-life scenario, making it possible to detect unexpected results and investigate the advantages and disadvantages of each MHS performed from various perspectives. The following scenarios are evaluated in order to achieve this objective and measure the effectiveness of the researched MHS:



(a) (b)

(c)

Figure 5.4: Implemented simulation in a 3D virtual environment. **(a)** FP randomly gathering around the breeding site. **(b)** FP starts searching trees. **(c)** A tree becomes pollinated.

- FP foraging generally begins in dark, moist places, such as decomposing banana trees, which also function as FP breeding grounds.
- The FP's beginning position is random and close to its breeding area.
- The experiment uses the following scenarios: the first case in which the number of trees available and the number of FP are the same; the second case in which there are more accessible trees than FP; and the third case in which there are more FP than trees available.

Table 5.1 and Figure 5.5 illustrate how the simulation operates in each experiment. For further details, can see the following, where explains how the simulation works in detail:

Table 5.1: Scenario of experiment.

Materials	Experiment 1	Experiment 2	Experiment 3
FP	10	10	20
Tree	10	15	15
Time (minutes)	20	20	20
Number of Simulation	20	20	20

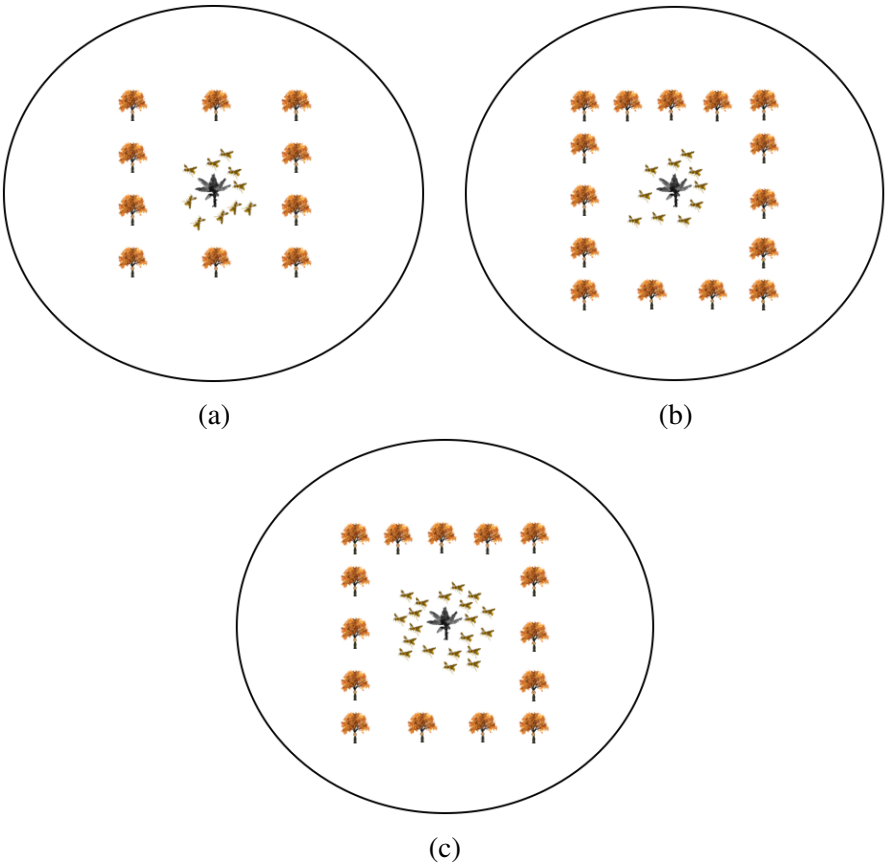


Figure 5.5: Search space environment of experiment. **(a)** Experiment 1. **(b)** Experiment 2. **(c)** Experiment 3.

1. The simulation was performed in a circular space with a diameter of 90 m consisting of FP, trees, and FP breeding sites in the middle of the circular space.

2. Figure 5.5 shows the location of each tree. Each experiment contained a different number of trees, and each tree was located about 6 to 10 meters apart.

3. The FP is placed at a random location within 5 m of the breeding site before the simulation begins and will remain at the same height during the simulation, never rising nor decreasing.

4. One of the algorithm methods is chosen, and FP will begin exploring trees in circle space for 20 minutes without crossing the border every time the simulation starts. The time steps for Idle-Jaya, Idle-Cuckoo, Lévy, and Defender Aggressor Game (DAG) is 1 second and for Idle-CSA (Crow Search Algorithm) is 2 seconds because when the condition for asking ”memory” needs another response from another FP.

5. Once an FP is within 3 meters of an unpollinated tree, the nectar level of the FP increases, and the FP carries pollen from that tree. The tree becomes pollinated if the FP delivers pollen from another tree to the unpollinated tree.

6. Each tree reveals a single flower. Once the FP is near a tree, it will collect one nectar. After 30 seconds, the flower will reproduce nectar and not produce flowers again if pollination is complete.

7. FP fly through in explorer of trees till a time limitation is reached.

5.3.2 Methodology of Idle Metaheuristic

In this part, we suggest an alternative approach for this experiment. The first method is from the Swarm Intelligence Algorithm, and we choose the Jaya Algorithm [71], Crow Search Algorithm [114], and Cuckoo Search Algorithm [2]. Second, the Individual Random Search method, as represented by Levy Flight [115]. The last method uses the Multi-Agent System Differential Search, i.e., the Defender Aggressor Game (DAG). We did not utilize the original algorithm in this experiment, but we adapted it to handle deferred fitness. This process means we will not perform direct fitness evaluation, which is necessary primarily for the Swarm Intelligence Algorithm.

The FP may also not focus on optimizing nectar consumption because this has no direct influence on the number of pollinated trees, nor can the tree supply any way to directly enhance the number of pollinated trees; this still requires the FP. On the other hand, the

Swarm Intelligence Algorithms can be modified to retain only their exploration components. Thus we employed them in ”idle-mode” for the exploration portion only; fitness value assessments were not used. This algorithm modification replaces all fitness values based on the internal processing of those algorithms with random decisions. Then in the following subsections, we will provide further explanation.

Swarm Intelligence Algorithm

We present three swarm methods in this chapter to solve the FP pollination problem. These algorithms, which were inspired by the behavior of social insect colonies and other animal societies, are known as Swarm Intelligence (SI) as a sub-discipline of MHS and are one of the most commonly utilized approaches by researchers to overcome complicated issues [116, 117, 118]. SI, in particular, is widely utilized as a source of inspiration in natural biological systems, including the collaborative investigation of the behavior of individuals from communities interacting with others locally. However, as previously stated, we adopt an “idle-mode” variation rather than direct fitness evaluations on SI.

It is important to remember that the algorithms perform in concurrent mode, with each FP mentioned being evaluated at a new position on a periodic schedule within the simulation. There is no looping across all FP individuals, and neither is the algorithm controlled centrally. Furthermore, all FP repositionings are typically constrained by not exceeding a maximum distance from the hives. Every pollination step in the following algorithm corresponds to the descriptions given above, and the pollinated tree is in the condition that the current FP is already carrying pollen collected from a different tree following the algorithm steps.

Method 1: Idle-Jaya

We have previously discussed the Jaya algorithm in this chapter 3. We chose this method since it has no parameters to configure and is recognized as a simple algorithm. As previously stated, the concept of Jaya’s algorithm tends towards the best and away from the worst; however, we altered the concept slightly for this experiment. The Idle-Jaya

Algorithm 5.1: The Idle-Jaya

Initialize each FP position
Initialize each Tree position
while termination condition not satisfied:
    $FP_{current}$ choose two another random FP; FP_1 and FP_2.
    Repeat until $FP_1 \neq FP_2$.
    Get position of FP;
    $X_{current} = FP_{current}$ position, $X_1 = FP_1$ position, $X_2 = FP_2$ position.
    Calculate distance FP; $d_1 = d(X_{current} - X_1)$, $d_2 = d(X_{current} - X_2)$
    if $d_1 > d_2$:
        $X_{far} = X_1$; $X_{near} = X_2$.
    else
        $X_{far} = X_2$; $X_{near} = X_1$.
    end if
    if $r > 0.5$ (chance to move to new position):
        Normalized vector; $X_{towards} = X_{far} - X_{current}$.
        Normalized vector; $X_{away} = X_{near} - X_{current}$.
        Calculate new position using; $X_{new} = X_{current} + (r * X_{towards}) - (r * X_{away})$.
        Update new position.
    end if
    Calculate distance between FP and nearest Tree; $d_{tree} = d(X_{new} - X_{tree})$.
    if $d_{tree} \leq 3$ ($T_{nearest}$ found):
        if FP carry poll of $T_{poll} \neq T_{nearest}$:
            Pollination.
        end if
        FP carry poll of $T_{nearest}$; $T_{poll} = T_{nearest}$
    end if
end while

concept is moved randomly towards the further FP and away from the closer FP in the algorithm below 5.1, which substitutes the idea of the best and worst individual in the regular Jaya basic euation (3.1) state in Chapter 3.

In this version 5.1, known as ”Idle-Jaya,” two additional FPs are chosen at random, with the closest designated as ”worst” and the furthest designated as ”best,” so that the FPs will instantly try to approach the other FP that is farthest away. However, when each other is adjacent to another FP, every FP will immediately keep away because it is detected as ”worst.” This condition makes it apparent that FP cannot explore together.

Method 2: Idle-CSA

Algorithm 5.2: The Idle-CSA

Initialize each FP position
Initialize each Tree position
Set reach of step; $reach$.
Set base position; $basepos$.
Set radius; $radius$.
while termination condition not satisfied:
    Choose another FP randomly.
    Get a response from that FP.
    $response = getResponse()$.
    $< case : receiveposition >$.
    Define target move; $m = response$
    Get current FP position; $X_{current}$.
    Calculate distance between target move and current FP position; d.
    Calculate new position; $X_{new} = X_{current} + (m - X_{current}) * r * d$
    Memorize position; $X_{memory} = X_{new}$
    Update new position.
    Calculate distance between FP and nearest Tree; $d_{tree} = d(X_{new} - X_{tree})$.
    if $d_{tree} \leq 3$ ($T_{nearest}$ found):
        if FP carry poll of $T_{poll} \neq T_{nearest}$:
            Pollination.
        end if
    end if
    $< case : giveposition >$.
    $getResponse()$:
        $X_{memory} = X_{current}$.
        if $r > awareness$.
            $response = X_{memory}$.
        else
            $response = basepos + r - radius$.
        end if
    end getResponse
end while

Since the crow is considered an intelligent bird, researchers Askarzade [114] suggested a study on optimizing the crow's hunt for food, called the Crow Search Algorithm (CSA).

The SI algorithm is adapted from the crow's habit of storing food in hidden locations and retrieving it when needed.

$$X^{i,iter+1} = X^{i,iter} + r_i.fl^{i,iter}(m^{j,iter} - X^{i,iter}) \tag{5.1}$$

Notation:

$X^{i,iter}$	Position of crow i at time $iter$ in search space;
r_i	Random number with uniform distribution between 0 and 1;
$fl^{i,iter}$	Denotes the flight length of crow i at iteration $iter$;
$m^{j,iter}$	Denotes either the position of hiding place of crow j at time $iter$ or a random new location in search space (crows' deception).

It is described in the algorithm of the Idle-CSA algorithm 5.2 that each FP can function as both a position giver and a position receiver simultaneously. There is also a "response" parameter in Idle-CSA that provides a sign to other FPs. This "response" parameter works similarly to the "memory" parameter in the standard CSA formula 5.1, but in Idle-CSA, the "response" parameter includes the previous memory value and also a random position within range.

Method 3: Idle-Cuckoo Search via Lévy Flights Algorithm

The cuckoo search is an algorithm that integrates various cuckoo species' breeding activities with Lévy's flying behavior. Cuckoo Search has two search modes: local and global search, both regulated by potential redirects. Consequently, the global search space can be investigated more efficiently, increasing the probability of discovering the global optimum. Because of this reason, local searches contribute to around one-quarter of total search time, whereas global searches contribute to three-quarters of overall search time [119]. The primary distinction between other SI algorithms as a SI algorithm is that FP implements various individual position updates to itself rather than other individuals, mimicking the cuckoo parasite's habit of laying its eggs in the nests of other birds.

In the formula (5.2), there is an entry-wise multiplication (Hadamard product) where value is derived from both matrices, which must have the exact dimensions. The matrix c

of the element-wise matrix product $a * b = c$ has the exact dimensions as dimensions a and b.

$$X_i^{(t+1)} = X_j^t + \alpha \bigoplus Lévy(\lambda) \tag{5.2}$$

Notation:

$X_i^{(t+1)}$	Generating new solutions $X^{(t+1)}$ for a cuckoo j; Lévy step is added to position of individual j;
$\alpha > 0$	The step size which should be related to the problem scale;
α	Weight factor of Lévy step;
$\bigoplus$	Entry-wise multiplications.

Algorithm 5.3: The Idle-Cuckoo Search via Lévy Flights Algorithm.

Initialize each FP position
Initialize each Tree position
Set reach of step; *reach*.
while termination condition not satisfied:
    $FP_{current}$ choose another one random FP, FP_1.
    Get current FP position; $X_{current}$.
    Get position of another FP; $X_1 = FP_1$ position.
    Calculate cuckoo position using Eq. (5.2); $X_{cuckoo} = X_1 + RandomLevy() * 0.1$.
    Calculate new position; $X_{new} = X_{current} + reach * X_{cuckoo}$
    Update new position
    Calculate distance between FP and nearest Tree; $d_{tree} = d(X_{new} - X_{tree})$.
    if $d_{tree} \leq 3$ ($T_{nearest}$ found):
        if FP carry poll of $T_{poll} \neq T_{nearest}$:
            Pollination.
        end if
        FP carry poll of $T_{nearest}$; $T_{poll} = T_{nearest}$
    end if
end while

Moreover, cuckoo searches are more efficient than ordinary random walks because global searches engage Lévy flights instead of standard random walks. We add a ”reach”

parameter to algorithm 5.3 that works as a multiplier to extend the Lévy Flight steps. We assign a value of reach = 5 to this experiment. The details of Lévy's flight will be described in the following algorithm 5.4, which implements the same principle. The pollination process will be pretty indolent if only the standard Cuckoo search algorithm is used, and FP will be unable to pollinate cocoa flowers since the steps done by the Lévy Flight are pretty limited. To avoid excessive FP shift, we adopt the suggested step size of 0.1. The following method, which employs the same idea, describes Lévy flight details.

Individual Random Search

Random search is a class of numerical optimization methods that do not need the gradient of the problem to be improved, allowing it to be utilized on neither continuous nor differentiable functions. Iteratively moving to better positions in the search space, sampled from a hypersphere surrounding the current position, is how random search works [120]. The method presented here is a local random search in which each iteration relies on the candidate solution from the previous iteration. Individual random searches are conducted by individuals who perform their random investigations without involving other FP locations.

Method 4: Lévy Flights

The Lévy flight is a random walk with an arbitrary stride length drawn from the Lévy distribution with infinite variance and mean. Numerous researchers have discovered that the Lévy technique is universal, and many modifications to improve the efficiency of Lévy flights have been created [121]. Furthermore, Lévy flights are widely recognized for solving optimization-related diffuseness, scaling, and transmission problems. According to Reynolds and Frye's study [122], fruit flies influence Lévy's or Drosophila melanogaster's flight-style irregular free-scale search patterns searching their surroundings by a sequence of straight flight paths punctuated by abrupt 90° turns.

As with Idle-Cuckoo, we add a "reach" parameter with the same value as Levy's step to this algorithm 5.4. This algorithm demonstrates that a random walk creates Levy flights with stride lengths selected from the stable levy distribution. A simple power-law formula is defined using the Levy probability distribution. A simple power-law formula is defined

Algorithm 5.4: Individual Lévy Flight.

Initialize each FP position
Initialize each Tree position
Set β.
Set list sigma value; $list\_sigma$.
Set reach of step; $reach$.
while termination condition not satisfied:
    if $0.5 < \beta < 1.95$:
        Calculate index; $i = (\beta/0.05) - 1$.
        Get sigma, $\sigma = list\_sigma[i]$.
        Choose random value; r_1, r_2.
        Calculate normal distribution 1; $nd_1 = \sqrt{log(r_1) * (-2)} * cos(2\pi r_2) * \sigma$
        Calculate normal distribution 2; $nd_2 = \sqrt{log(r_1) * (-2)} * cos(2\pi r_2)$
        Calculate random Lévy; $levy = nd_1/|nd_2|^{(1/\beta)}$
    end if
    Calculate new position; $X_{new} = levy * reach * 0.1$
    Update new position.
    Calculate distance between FP and nearest Tree; $d_{tree} = d(X_{new} - X_{tree})$.
    if $d_{tree} \leq 3$ ($T_{nearest}$ found):
        if FP carry poll of $T_{poll} \neq T_{nearest}$:
            Pollination.
        end if
        FP carry poll of $T_{nearest}$; $T_{poll} = T_{nearest}$
    end if
end while

using the Levy probability distribution. The index of Levy distribution is $0 < \beta \leq 2$ [123] as shown in the equation (5.3).

$$Lévy(s) \sim |S|^{-1-\beta} \tag{5.3}$$

Multi-Agent System

A multi-agent system (MAS) is an agent that operates as a system composed of multiple interacting computer components. MAS is considered a suitable nature metaphor for comprehending and building various sorts of artificial social systems. The MAS idea does not

appear to be confined to a particular application domain but instead appears to be standard across many application domains [124]. Although MAS is frequently used to simulate self-regulating systems and emergent behavior, it has not been generally used for a random search and immediate solutions to optimization problems in a generic sense. Pollination Problems also has MAS application potential since it is not connected to the direct evaluation of fitness functions.

Method 5: Defender-Aggressor-Game (DAG)

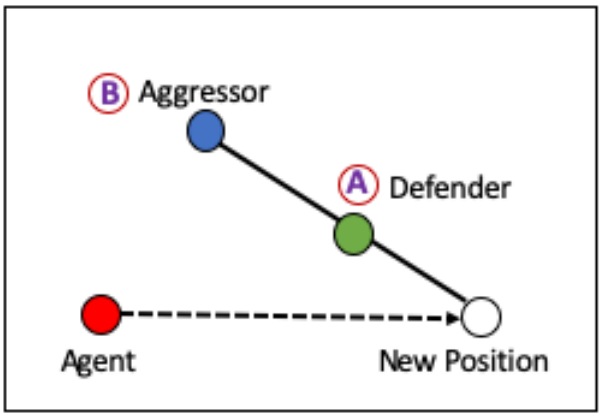


Figure 5.6: Rules for playing the Defender Aggressor Game.

We use a simple participation game based on the Defender-Aggressor-Game (DAG), in which each participant selects two other players at random. As illustrated in Figure 5.6, we suppose that the chosen players are A and B. In this game, everyone attempts to position themselves so that their A (the ”Defender”) player is constantly between them and their particular B (the ”Aggressor” player). Everyone in this game aims to arrange themselves with A and B at the exact time. This basic rule keeps the dynamic constant with all agents moving randomly, with a low possibility that the pattern will settle to a line-like arrangement of all participants [125].

As previously explained, for participation games, we add two parameters, ”aggressor” and ”defender,” to this algorithm 5.5. FP will be randomly selected as players A (defender) and B (aggressor) and then multiplied by the safety factor to generate a new position. The safety factor value is 1.2, and this parameter allows FP to move farther behind the defender

Algorithm 5.5: DAG

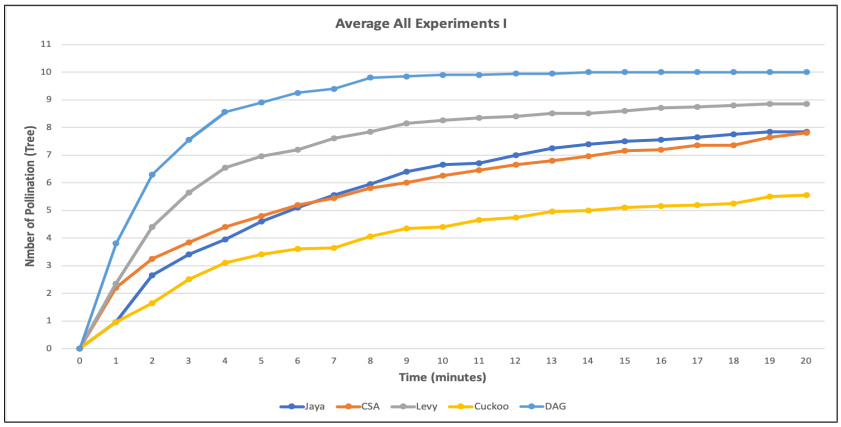
Initialize each FP position
Initialize each Tree position
while termination condition not satisfied:
    $FP_{current}$ choose two another random FP; FP_1 and FP_2.
    Repeat until $FP_1 \neq FP_2$.
    Get position of FP; $X_1 = FP_1$ position, $X_2 = FP_2$ position.
    FP_1 as aggressor, FP_2 as defender.
    $X_{new} = X_1 + (X_2 - X_1) * 1.2$
    Update new position.
    Calculate distance between FP and nearest Tree; $d_{tree} = d(X_{new} - X_{tree})$.
    if $d_{tree} \leq 3$ ($T_{nearest}$ found):
        if FP carry poll of $T_{poll} \neq T_{nearest}$:
            Pollination.
        end if
        FP carry poll of $T_{nearest}$; $T_{poll} = T_{nearest}$
    end if
end while

but not far enough. Values more than one but less than one are typical alternatives.

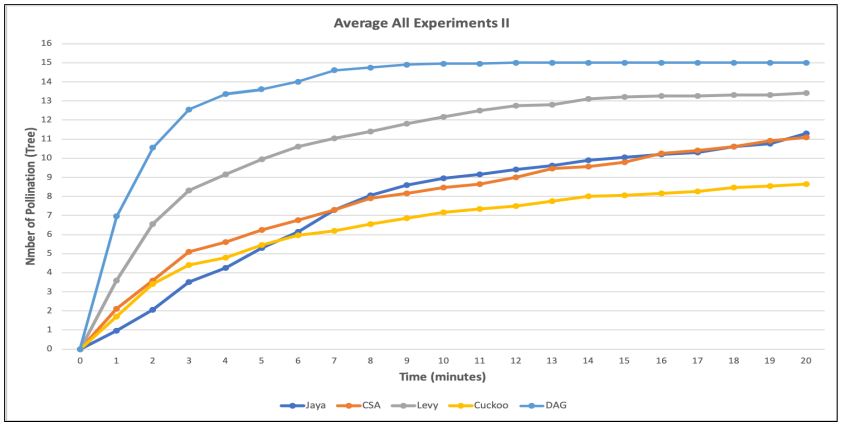
5.4 Experimental Evaluations

This section introduce the experimental results and the impact of FP behavior after implementing the metaheuristic algorithm approach from the three different techniques. The differences are demonstrated in Figure 5.7 by showing the average pollination of trees. The time shown on the x-axis is in the range from 0 to 20 minutes. Pollination occurs on average between 0:00 and 0.59, according to the first minute of the graph. DAG pollinated more trees in the first minute, with a mean value of 3.8, as shown in Figure 5.7a. Lévy comes in second with an average score of 2.35, followed by Idle-CSA 2.2, Idle-Cuckoo 2.2, and Idle-Jaya 0.95.

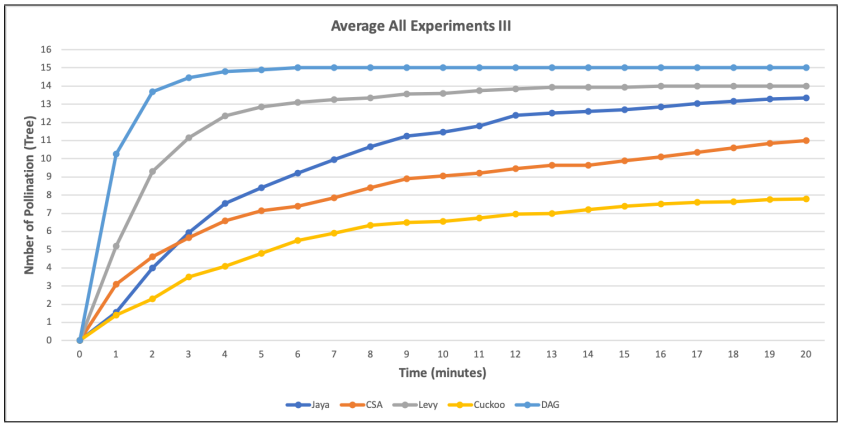
Overall simulations revealed significant variations in results when swarm algorithms, such as Idle-Jaya, Idle-CSA, and Idle-Cuckoo, were used, compared to searches that did



(a)



(b)



(c)

Figure 5.7: The average results from all simulation. **(a)** Experiment I. **(b)** Experiment II. **(c)** Experiment III.

not utilize swarm behavioral method, such as Lévy flight and DAG from the average percentage results. Swarm search algorithms like Idle-Jaya, Idle-CSA, and Idle-Cuckoo need longer time pollinate more trees, whereas Lévy and DAG flights pollinate more trees in less time. Since this algorithm is intended to operate together in the search, this simulation indicates that the swarm methodology in pollination here needs a longer duration of time. However, because the range of each FP varies, more trees are pollinated in a shorter amount of time when utilizing multi-agent system search and individual random search method. FP operates independently without the need to interact with others FP. In other words, when we use the SI method, we can see that the FP swarms are all clustered around the same tree, where they can have all of the nectar but will not cross-pollinate before flying to another tree. The nectar will no longer be available to other FPs once the first FP successfully pollinated the tree.

Figure 5.7 represents the average result of the tree pollinated by FP based on time, whereas Table 5.2 illustrates how much nectar the FP attempted to collect while exploring the tree. It can be seen in Table 5.2 that the percentage value of Lévy flight and DAG in experiment 2 is higher than in experiment 3, indicating that when there are more trees than FPs, almost every FP can find nectar; however, when there are more FPs than trees, some FPs are unable to obtain nectar.

Table 5.2: The average nectar amount from all simulations.

Algorithm	**Experiment 1**	**Experiment 2**	**Experiment 3**
Idle-Jaya	9.55	12.10	15.50
Idle-CSA	10.10	13.20	13.95
Idle-Cuckoo	9.00	15.10	10.40
Lévy flight	10.50	18.80	15.50
DAG	11.70	17.70	16.80

The Gini Index is a well-known concentration index developed more than a century ago by Corrado Gini [126] to assess the degree of disparity in income and wealth distribution. The Gini index is the single best indicator of inequality, according to Morgan and researchers [127]. Therefore in this simulation, we used the Gini Index to represent whether FP nectar intake was distributed evenly and fairly among the FP population, as opposed to a situation in which certain FPs consumed the most of the nectar on their own. Table 5.3

shows the Gini index results for FP. The Gini index of all experiments indicated that FP distributes nectar equitably since zero signified perfect equality between FPs and one indicated maximal inequality between FPs; other than that, all values were equal. However, the advantages of DAG over other methods stand out in this case.

Table 5.3: The Gini Index average of the simulated representation of FP distribution nectar.

Algorithm	**Experiment 1**	**Experiment 2**	**Experiment 3**
Idle-Jaya	0.53	0.47	0.63
Idle-CSA	0.47	0.39	0.55
Idle-Cuckoo	0.49	0.40	0.64
Lévy flight	0.48	0.37	0.58
DAG	0.41	0.31	0.49

5.5 Discussion

The conclusion we can get after modeling cacao flower pollination scenarios using different random search algorithms was that Lévy Flight and DAG outperformed selected SI search method, especially Idle-Jaya, Idle-CSA, and Idle-Cuckoo Search (which also incorporates Lévy Flight). In the first several minutes of the three case simulations presented, Lévy and DAG flights pollinate more trees than Idle-Jaya, Idle-CSA, and Idle-Cuckoo. When comparing search algorithms to neural architecture search (NAS), the article [128] finds that the evolutionary algorithm effectively optimizes NAS. Instead, in the case of Pollination Problems, a random search may be faster, but it does not guarantee the best results.

This experiment also reveals that Idle-MH performs a random search like DAG and Levy, but the difference is that the search procedure uses single or independent searches and searches with population or swarm. This difference makes us wonder why the value of the results differs so significantly. Further study is required to determine the reasons underlying these findings.

The DAG method presented here is not that dissimilar to the standard evolutionary algorithm. This state is related to a specific case of DE. The following summarizes the classical DE procedure: A basic differential mutation procedure resulting from two distinct individuals chosen from the population disrupts a randomly selected individual as the base

vector. Then, to identify whether individuals survive, produce progeny candidates using a one-to-one selection approach [129]. DE adds a vector of differences between two other individuals, or a randomly chosen third place is the main point for a new search candidate position. DE adds a difference vector between two other individuals, or a randomly chosen third place is the primary point for a new search candidate position, while DAG accomplishes the same by adding a Defender and Aggressor difference vector to Defender. Thus, we assume that DAG is a differential algorithm as well. DE is well-known for being a solid MHS across various application cases while remaining simple to use and implement. DE is also the preferred algorithm for addressing continuous real-valued optimization problems [63, 130, 131]. Based on this assumption, we can conclude that DAG outperforms other Idle-SI algorithms since it follows the same logic as DE.

Aside from DAG, the Lévy flight performed well, coming in second place following DAG. These findings support Kishor Kisan Ingle and Ravi Kumar Jatoth's [77] suggestion to integrate Lévy flights into Jaya's base algorithm to increase exploration and exploitation capabilities throughout the search phase. This study recommends that Lévy flights be included in Jaya's base algorithm to allow global search in the early and local phases of the final probe, therefore increasing the algorithm's exploration and local optima avoidance capabilities. However, Idle-Cuckoo Search, the SI method that originated from Cuckoo Search, already involves Lévy flights, but the findings indicate the lowest performance of all the algorithms investigated here.

What are the primary advantages of such a study? Furthermore, we cannot include head-to-head competition between algorithms, as is typical in other studies of metaheuristic algorithm application, such as [132]. This is merely owing to the deficiency of a fitness function that is immediately available. Even when those algorithms are reduced to their exploratory component, there are substantial disparities in the congruent objectives of tree pollination contemporaneous with nectar collection. We can see that three different kinds of algorithms related to the more general situation of random search strategies:

- Individual strategies, such as Lévy flight, allow each FP to pursue its trajectory independently of the other FPs. This argument also is well supported by biological knowledge of insect flying patterns. We can confirm that it is a good strategy but not the most outstanding quality method.

- When determining the next move, the FP refers to one or more other FP positions. This is generally implemented in all Swarm Intelligence algorithms. However, we can see no substantial improvements in nectar consumption or an increase in the number of pollinated trees. From a biological standpoint, it is similarly based on the assumption that insects can distinguish their species amid other items in their environment.

- A differential approach where the FP determines the next step depending on the offset between two objects in the environment. Our study clearly shows the benefits of this strategy: it pollinates the most significant number of trees while also ensuring the most equitable distribution of nectar in the population. Moreover, the pun is on "different objects" and not necessarily other FP to take as reference. This implies that the same method may work well, for example, using other insect species as a point of comparison. This method is subject to further investigations.

In practice, the promotion of diverse techniques can encourage pollination, which will necessitate agricultural experiments. Another influence is on the analysis of related optimization issues in this category of deferred fitness problems. In reality, we may discover many problems previously challenging to approach, such as the evolution of parasitism as an example of biology. Furthermore, there are corresponding restrictions throughout the food supply chain, such as manufacturing agricultural goods arrive in a new state at several consumer locations.

As a restriction on the proposed methodlogy scalability can be seen. While there are no significant differences in relative number of pollinated trees by either increasing the number of trees or the number of FP, and also there is no reason to change scale here (consider limited range of FP), the analysis can not be extended to higher dimensions. The search effort will increase exponentially with dimensions as stated by the famous Curse of Dimensionality.

5.6 Summary

In this chapter, we investigate a pollination optimization problem that also happens to be an optimization problem with a deferred fitness evaluation. FP and tree interact in a smooth but also contingent manner to accomplish this purpose.

We investigated three different search methods: fitness-free versions of Swarm Intelligence Algorithm, i.e., Idle-Jaya, Idle-CSA, and Idle-Cuckoo Search; the Individual Random Search method, i.e., Lévy flight; and finally, the Multi-Agent Systems search method, i.e., Defender-Aggressor-Game (DAG). These methods were evaluated in order to simulate cocoa flower pollination.

We chose cacao pollination as the simulation scenario since the cacao flower differs from other flowers. The distinction is their small size (maximum diameter 3 cm), which allows only tiny insects to pollinate them. Aside from that, self-pollination of inappropriate types will not result in adequate fertilization; thus, cross-pollination is the only way to ensure successful fertilization.

From the results of this investigation, we can recognize the differences between the random search strategies. The DAG differential method has significant advantages in terms of the main aim and ratio of pollinated trees. There was no significant difference in nectar consumption or distribution, but there was a tendency toward better DAG values. In summary, we may infer that there is no best method in every situation. However, the findings of these simulations for multi-agent and random search may be more relevant for pollination cocoa in the real world. The simulation approach is also intended to assist farmers with inadequate pollination of cocoa flowers. The relevant settings and experiments may also be carried out by farmers using the same simulation software to study the influence of various factors on pollination, increasing cocoa plant productivity. Through utilizing this simulation, farmers may use this simulation to manage or design the location of their trees or plants, as well as the sequence of potential nests or breeding grounds for pollinator nests. Besides, solve the problem of cocoa pollination, this simulation approach can also contribute to the types of flowers that perform cross-pollination, such as apples, pumpkins, daffodils, grasses, maple trees, and most flowering plants. The whole methodology actually refers to interlocked ecosystems, having pollinator and flowers/tree here, and thus can

be considered to also study other such systems: parasite and host, mycelium and plants, to name two currently hot research topics.

Chapter 6

Conclusion and Future Works

The metaheuristic search (MHS) method has been widely implemented to various engineering problems, and many experts attempt to enhance its performance to address new and complex problems. Since MHS algorithms solve various problems in many areas, they have become one of the most promising research areas. Therefore, the primary objective of this research is to recommend novel application fields through the use of MHS algorithms. We propose solving optimization problems in fields of real-world problems where MHS have not yet solve them. So, to solve this problem, we utilize the problem model as an application to solve the problem indirectly. In conclusion, we chose the title incubation of MHS algorithms, where ”incubation” refers to the process of developing MHS algorithms into a new application field out of their typical application modality, i.e. optimization..

The first approach is to conduct parametric research on MHS to understand better how parameters are used. In this study, we were using the Jaya algorithm since it is parameter-free and similar to PSO. Most known swarm-based algorithms include general controlling parameters, such as particle number, or particular parameters, such as weights, personal best, and globals best in the PSO algorithm, but this is different in Jaya's method, which lacks a general controller parameter. Therefore, we introduce weights to denote equal weights on both the best and worst sides. Furthermore, the second best and second worst impacts were investigated to understand better how the algorithm operates. According to the findings of this investigation, Jaya still performs better without weighting parameters for the majority of functions.

The second approach is to incorporate MHS for Blockchain Proof of Work (PoW) to address wasteful energy consumption in bitcoin usage. Utilizing the Traveling Salesperson Problem as a model problem, we propose a concept for using iterative optimization techniques to generate the PoW required to add a block to the Blockchain. The experimental results showed that using the suggested concept in a real-world scenario can create robust new schemes.

The third approach is to provide an Idle-Metaheuristic for flower pollination simulation. We examined three random search methods: fitness-free versions of the Swarm Intelligence Algorithm, i.e., Idle-Jaya, Idle-CSA, and Idle-Cuckoo Search; the Individual Random Search method, i.e., Lévy flight; and, finally, the Multi-Agent Systems search method, i.e., Defender-Aggressor-Game (DAG). These approaches were evaluated in order to imitate the cocoa flower pollination scenario. We adopted cocoa pollination as the simulation scenario because cocoa flowers are distinct from those of other plants. Since they are tiny (maximum the diameter of the cacao flowers is 3 cm), only little insects can pollinate them. We can see the differences between random search techniques based on the findings of this investigation. There is an obvious advantage from the differential approach DAG in terms of the main aim and the ratio of pollinated trees. There are no substantial differences in nectar intake and distribution, although DAG has a higher value tendency.

Based on the results presented in each chapter, it is clear that each method used in this study provides a different set of optimization solutions. We cannot expect them always to discover the ideal solution, but we can expect them to find a relatively decent or even optimal solution most of the time, and more crucially, in a reasonable amount of time. Likewise, with several methods not utilized in this study, this does not imply that the algorithm is worst; however, we have attempted some in previous experiments, and the intended outcome was not obtained. Modern MHS algorithms almost always perform effectively for a wide variety of complex optimization tasks. However, the MHS method has limitations in some optimization problems. Wolpert and Macready showed in 1997 [133] that if algorithm A outperforms algorithm B for some problems, then B will outperform A for other problems. That is, there are no universally effective algorithms. The primary goal of optimization and algorithm development research is to develop or select the best appropriate and efficient algorithm for a particular optimization problem.

Furthermore, after conducting the experiments in this study, we could answer the research question stated in Chapter 1.

1. The study findings indicate that the MHS algorithms could also be utilized to solve other practical problems like in Chapter 4, which gives a solution to iteratively implementing MHS methods to provide the PoW required to extend the Blockchain with new blocks. Furthermore, in Chapter 5, the MHS algorithms solves a problem in a new field inspired by nature, mainly the pollination of cacao flowers.

2. The MHS algorithms can be easily adjusted for a specific application, as evidenced by the results obtained in this study to solve problems in Blockchain and Cocoa Flower Pollination Simulations. As we did in the Jaya algorithm discussed in Chapter 3, we added parameters to the Jaya standard, but the results showed that the Jaya standard performs better without adding another parameter. Therefore, in Cocoa Flower Pollination Simulation, we use the Jaya algorithm further, but instead of adding parameters, we adjust the concept to adapt to the FP search.

3. As described in Chapter 5, demonstrates that the MHS algorithms can be modified for use in other application domains. We propose an ”idle-mode” SI algorithm for exploration without evaluating fitness values. We modify the algorithm to substitute all fitness value-based internal processing with arbitrary decisions.

The three approaches also provide knowledge about how to improve the effectiveness of the MHS algorithms in overcoming different fields by setting parameters as efficiently as possible according to the problem to be solved. These three approaches indicate that implementing MHS algorithms can solve the problem in this study by indirectly solving the optimization problem through the concept of the same problem model. In addition, the results of the study of these three approaches also show that parameter settings in the MHS algorithms are not so difficult to use, and each parameter can be adjusted to solve the real-world problem. This study is expected to assist other researchers in improving and developing the performance of MHS algorithms used to deal with other real-world difficulties.

The conclusions of the optimization utilizing the MHS algorithms here indicate that our recommended methods are efficient and promising; however, the work in each research approach was only partially completed, and there are possibilities worth studying. The following are a few relevant research topics that can be explored for future study.

- We must continue to delve into the Jaya algorithm (one of the MHS algorithms we chose to study parametric studies). This algorithm requires further research because the original statement "towards the best and away from the worst" may not tell the whole story leaving some assumptions unaddressed. In addition, we would like to investigate by using the Sugeno measure for weight assignment and compare it to the Jaya algorithm to consider the influence of the weights and the parameters from these two method. The Sugeno measure is a non-additive measure that is a significant generalization of the probability measure developed by Sugeno [134] in 1974. Since the Sugeno measure frequently solves situations requiring a minimum and maximum weighting [135, 136, 137], we consider implementing it and comparing it with the Jaya algorithm.

- In Blockchain issues, we only use PSO and QPSO and do not try other heuristic and MHS algorithms to optimize it. We can attempt another algorithm approach and compare it for further insights on this issue.

- As mentioned in Chapter 5, there is a significant difference in results between the random search and Idle-MH methods adopted, and further research is needed to determine the specific explanation for the difference in this MHS procedure. In addition, we must continue to explore the EC and SI methods to see more possible findings in the future.

- After implementing a MHS algorithms on pollinating cocoa flowers, we can make additional changes to be more advantageous to farmers by enhancing the post-pollination process, specifically the production of cocoa beans that will be supplied to customers. In addition, we considered implementing the Taguchi [138] method in the development of this study. The Taguchi method is used in the development of this study to design experiments and determine different parameter settings of each algorithm that

produce the best quality characteristics (performance measures) to achieve the goal of producing high-quality products at low costs for manufacturers.

Bibliography

[1] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

[2] Xin-She Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World Congress on Nature & Biologically Inspired computing (NaBIC)*, pages 210–214. Ieee, 2009.

[3] Seyed Jalaleddin Mousavirad and Hossein Ebrahimpour-Komleh. Human mental search: a new population-based metaheuristic optimization algorithm. *Applied Intelligence*, 47(3):850–887, 2017.

[4] Kuk-Hyun Han, Kui-Hong Park, Ci-Ho Lee, and Jong-Hwan Kim. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In *Proceedings of The 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 2, pages 1422–1429. IEEE, 2001.

[5] Jingan Yang and Yanbin Zhuang. An improved ant colony optimization algorithm for solving a complex combinatorial optimization problem. *Applied Soft Computing*, 10(2):653–660, 2010.

[6] Amanpreet Kaur and Bikrampal Kaur. Load balancing optimization based on hybrid heuristic-metaheuristic techniques in cloud environment. *Journal of King Saud University-Computer and Information Sciences*, 2019.

[7] Vasanth Sarathy. Real world problem-solving. *Frontiers in Human Neuroscience*, 12:261, 2018.

[8] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of Combinatorial Optimization*, pages 2093–2229. Springer, 1998.

[9] Alan M Turing. Turing's treatise on enigma. *Unpublished Manuscript*, 1940.

[10] John Holland. Adaptation in artificial and natural systems. *Ann Arbor: The University of Michigan Press*, page 232, 1975.

[11] Kenneth ALAN de JONG. Analysis of the behavior of a class of genetic adaptive systems. *Technical Report No. 185, Department of Computer and Communication Sciences, University of Michigan*, 1975.

[12] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[13] Fred Glover. Tabu search and adaptive memory programming—advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Springer, 1997.

[14] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*, 1992.

[15] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT Press, 1992.

[16] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[17] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60–68, 2001.

[18] Sunil Nakrani and Craig Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12(3-4):223–240, 2004.

[19] DT Pham, A Ghanbarzadeh, E Koc, S Otri, S Rahim, and M Zaidi. The bees algorithm. *Technical Note, Manufacturing Engineering Centre, Cardiff University, UK*, 2005.

[20] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer, 2005.

[21] Xin-She Yang. Firefly algorithms for multimodal optimization. In *International Symposium on Stochastic Algorithms*, pages 169–178. Springer, 2009.

[22] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[23] Fred Glover, Manuel Laguna, and Rafael Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[24] S Gholizadeh and H Barati. A comprative study of three metaheuristics for optimum design of trusses. 2012.

[25] V Haji Haji and Concepción A Monje. Fractional order fuzzy-pid control of a combined cycle power plant using particle swarm optimization algorithm with an improved dynamic parameters selection. *Applied Soft Computing*, 58:256–264, 2017.

[26] Saman M Almufti. Historical survey on metaheuristics algorithms. *International Journal of Scientific World*, 7(1):1, 2019.

[27] Prakash Shelokar, Abhijit Kulkarni, Valadi K Jayaraman, and Patrick Siarry. Metaheuristics in process engineering: a historical perspective. In *Applications of Metaheuristics in Process Engineering*, pages 1–38. Springer, 2014.

[28] Anton Dekkers and Emile Aarts. Global optimization and simulated annealing. *Mathematical Programming*, 50(1):367–393, 1991.

[29] Wikipedia. Simulated annealing. Available at: `https://en.wikipedia.org/wiki/Simulated_annealing`. Accessed: 25-October-2021.

[30] Stephen M Goldfeld, Richard E Quandt, and Hale F Trotter. Maximization by quadratic hill-climbing. *Econometrica: Journal of the Econometric Society*, pages 541–551, 1966.

[31] Y Xin-She. An introduction with metaheuristic applications. *Engineering Optimization*, 2010.

[32] Wen-Chyuan Chiang and Robert A Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997.

[33] Wen-Chyuan Chiang. The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research*, 77:209–227, 1998.

[34] Eugenia Díaz, Javier Tuya, and Raquel Blanco. Automated software testing using a metaheuristic technique based on tabu search. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, pages 310–313. IEEE, 2003.

[35] Rafael Caballero, Mercedes González, Flor Ma Guerrero, Julián Molina, and Concepción Paralera. Solving a multiobjective location routing problem with a metaheuristic based on tabu search. application to a real case in andalusia. *European Journal of Operational Research*, 177(3):1751–1763, 2007.

[36] Wikipedia. Tabu search. Available at: `https://en.wikipedia.org/wiki/Tabu_search`. Accessed: 25-October-2021.

[37] Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.

[38] Roberto Battiti and Marco Protasi. Reactive search, a history-sensitive heuristic for max-sat. *Journal of Experimental Algorithmics (JEA)*, 2:2–es, 1997.

[39] Chih-Chin Lai and Chuan-Yu Chang. A hierarchical evolutionary algorithm for automatic medical image segmentation. *Expert Systems with Applications*, 36(1):248–259, 2009.

[40] Yangyang Li, Licheng Jiao, Ronghua Shang, and Rustam Stolkin. Dynamic-context cooperative quantum-behaved particle swarm optimization based on multilevel thresholding applied to medical image segmentation. *Information Sciences*, 294:408–422, 2015.

[41] Mohammad-H Tayarani-N, Xin Yao, and Hongming Xu. Meta-heuristic algorithms in car engine design: A literature survey. *IEEE Transactions on Evolutionary Computation*, 19(5):609–629, 2014.

[42] S Mousavi Rad, F Akhlaghian Tab, and K Mollazade. Application of imperialist competitive algorithm for feature selection: A case study on bulk rice classification. *International Journal of Computer Applications*, 40(16):41–48, 2012.

[43] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. Handbook of evolutionary computation. *Release*, 97(1):B1, 1997.

[44] David E Goldberg and John Henry Holland. Genetic algorithms and machine learning. 1988.

[45] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[46] Xiao-Feng Xie, Wen-Jun Zhang, and Zhi-Lian Yang. Social cognitive optimization for nonlinear programming problems. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 2, pages 779–783. IEEE, 2002.

[47] Holland John. Holland. genetic algorithms. *Scientific American*, 267(1):44–50, 1992.

[48] Kornel Chromiński and Magdalena A Tkacz. Epigenetic modification of genetic algorithm for outlier detection. *Procedia Computer Science*, 192:4178–4185, 2021.

[49] Hongxing Yang, Wei Zhou, Lin Lu, and Zhaohong Fang. Optimal sizing method for stand-alone hybrid solar–wind system with LPSP technology by using genetic algorithm. *Solar Energy*, 82(4):354–367, 2008.

[50] Alexander Von Homeyer. Evolutionary algorithms and their applications in chemistry. *Handbook of Chemoinformatics: From Data to Knowledge in 4 Volumes*, pages 1239–1280, 2003.

[51] Alex A Freitas. A review of evolutionary algorithms for data mining. *Data Mining and Knowledge Discovery Handbook*, pages 371–400, 2009.

[52] Hiroaki Imade, Ryohei Morishita, Isao Ono, Norihiko Ono, and Masahiro Okamoto. A grid-oriented genetic algorithm framework for bioinformatics. *New Generation Computing*, 22(2):177–186, 2004.

[53] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[54] D Nagesh Kumar and M Janga Reddy. Ant colony optimization for multi-purpose reservoir operation. *Water Resources Management*, 20(6):879–898, 2006.

[55] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[56] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[57] Gang Xu and Guosong Yu. Reprint of: On convergence analysis of particle swarm optimization algorithm. *Journal of Computational and Applied Mathematics*, 340:709–717, 2018.

[58] Weiyi Qian and Ming Li. Convergence analysis of standard particle swarm optimization algorithm and its improvement. *Soft Computing*, 22(12):4047–4070, 2018.

[59] Youness Khourdifi and Mohamed Bahaj. Heart disease prediction and classification using machine learning algorithms optimized by particle swarm optimization and ant

colony optimization. *International Journal of Intelligent Engineering and Systems*, 12(1):242–252, 2019.

[60] Chongchong Qi, Andy Fourie, and Qiusong Chen. Neural network and particle swarm optimization for predicting the unconfined compressive strength of cemented paste backfill. *Construction and Building Materials*, 159:473–478, 2018.

[61] Feng Wang, Heng Zhang, Kangshun Li, Zhiyi Lin, Jun Yang, and Xiao-Liang Shen. A hybrid particle swarm optimization algorithm using adaptive learning strategy. *Information Sciences*, 436:162–177, 2018.

[62] Shuyuan Yang, Min Wang, et al. A quantum particle swarm optimization. In *Proceedings of The 2004 Congress on Evolutionary computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 320–324. IEEE, 2004.

[63] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[64] Fahui Gu, Kangshun Li, Lei Yang, and Yan Chen. Combinatorial optimization of multi-agent differential evolution algorithm. *The Open Cybernetics & Systemics Journal*, 8(1), 2014.

[65] Jinping Wu and Siling Feng. Improved biogeography-based optimization for the traveling salesman problem. In *2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, pages 166–171. IEEE, 2017.

[66] Swagatam Das, Ajith Abraham, and Amit Konar. Automatic clustering using an improved differential evolution algorithm. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):218–237, 2007.

[67] Paolo Rocca, Giacomo Oliveri, and Andrea Massa. Differential evolution as applied to electromagnetics. *IEEE Antennas and Propagation Magazine*, 53(1):38–49, 2011.

[68] Ali R Yildiz. A new hybrid differential evolution algorithm for the selection of optimal machining parameters in milling operations. *Applied Soft Computing*, 13(3):1561–1566, 2013.

[69] Sultan Noman Qasem and Siti Mariyam Shamsuddin. Memetic elitist pareto differential evolution algorithm based radial basis function networks for classification problems. *Applied Soft Computing*, 11(8):5565–5581, 2011.

[70] Bin Qian, Ling Wang, De-xian Huang, Wan-liang Wang, and Xiong Wang. An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers & Operations Research*, 36(1):209–233, 2009.

[71] R Rao. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 7(1):19–34, 2016.

[72] RV Rao and VD Kalyankar. Parameters optimization of advanced machining processes using tlbo algorithm. *EPPM, Singapore*, 20:21–31, 2011.

[73] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[74] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, pages 65–70, 1979.

[75] R Rao and Vivek Patel. Comparative performance of an elitist teaching-learning-based optimization algorithm for solving unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 4(1):29–50, 2013.

[76] Kunjie Yu, Boyang Qu, Caitong Yue, Shilei Ge, Xu Chen, and Jing Liang. A performance-guided jaya algorithm for parameters identification of photovoltaic cell and module. *Applied Energy*, 237:241–257, 2019.

[77] Kishor Kisan Ingle and Ravi Kumar Jatoth. An efficient jaya algorithm with lévy flight for non-linear channel equalization. *Expert Systems with Applications*, 145:112970, 2020.

[78] Xin-She Yang and Xingshi He. Firefly algorithm: recent advances and applications. *International Journal of Swarm Intelligence*, 1(1):36–50, 2013.

[79] Momin Jamil and Xin-She Yang. A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2):150–194, 2013.

[80] MA Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

[81] A.-R. Hedar. Global optimization test problems. Available at: `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm`. Accessed: 8-December-2021.

[82] Chan-Jin Chung and Robert G Reynolds. Caep: An evolution-based tool for real-valued function optimization using cultural algorithms. *International Journal on Artificial Intelligence Tools*, 7(03):239–291, 1998.

[83] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605–1614, 2007.

[84] HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[85] Sonja Surjanovic and Derek Bingham. Virtual library of simulation experiments: Test functions and datasets. Available at: `https://www.sfu.ca/~ssurjano/index.html`. Accessed: 8-December-2021.

[86] AA Goldstein and JF Price. On descent from local minima. *Mathematics of Computation*, 25(115):569–574, 1971.

[87] Warid Warid, Hashim Hizam, Norman Mariun, and Noor Izzri Abdul-Wahab. Optimal power flow using the Jaya algorithm. *Energies*, 9(9):678, 2016.

[88] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[89] D Taylor. An analysis of bitcoin and the proof of work protocols energy consumption. *Growth, Impact and Sustainability*, 2018.

[90] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford University, 1979.

[91] Joost De Kruijff and Hans Weigand. Understanding the blockchain using enterprise ontology. In *International Conference on Advanced Information Systems Engineering*, pages 29–43. Springer, 2017.

[92] Imran Bashir. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained*. Packt Publishing Ltd, 2018.

[93] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoining: A scalable blockchain protocol. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 45–59, 2016.

[94] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[95] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *Self-Published Paper, August*, 19(1), 2012.

[96] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On security analysis of proof-of-elapsed-time (poet). In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 282–297. Springer, 2017.

[97] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Sonference on Computer and Communications Security*, pages 3–16, 2016.

[98] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE International Congress on Big data (BigData Congress)*, pages 557–564. IEEE, 2017.

[99] Yongquan Zhou, Qifang Luo, Huan Chen, Anping He, and Jinzhao Wu. A discrete invasive weed optimization algorithm for solving traveling salesman problem. *Neurocomputing*, 151:1227–1236, 2015.

[100] Mostafa Mahi, Ömer Kaan Baykan, and Halife Kodaz. A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing*, 30:484–490, 2015.

[101] Jun Sun, Wei Fang, Xiaojun Wu, Vasile Palade, and Wenbo Xu. Quantum-behaved particle swarm optimization: analysis of individual particle behavior and parameter selection. *Evolutionary Computation*, 20(3):349–393, 2012.

[102] Isaac López-López, Guillermo Sosa-Gómez, Carlos Segura, Diego Oliva, and Omar Rojas. Metaheuristics in the optimization of cryptographic boolean functions. *Entropy*, 22(9):1052, 2020.

[103] JA Winder. Recent research on insect pollination of cocoa. *Cocoa Growers Bulletin*, 1977.

[104] Gregor Claus, Wouter Vanhove, Patrick Van Damme, and Guy Smagghe. Challenges in cocoa pollination: the case of côte d'ivoire. *Pollination in Plants*, 39, 2018.

[105] SC BGRLBND. Studies in cacao: Part i. the method of pollination. *Annals of Applied Biology*, 12(4):403–409, 1925.

[106] GA Jones. The structure and pollination of the cacao flower. *West Indian Bull*, 12:347–350, 1912.

[107] Tohko Kaufmann. Behavioral biology of a cocoa pollinator, forcipomyia inornatipennis (diptera: ceratopogonidae) in Ghana. *Journal of The Kansas Entomological Society*, pages 541–548, 1974.

[108] Kofi Frimpong-Anin, Michael K Adjaloo, Peter K Kwapong, and William Oduro. Structure and stability of cocoa flowers and their response to pollination. *Journal of Botany*, 2014.

[109] Xin-She Yang. Flower pollination algorithm for global optimization. In *International Conference on Unconventional Computing and natural Computation*, pages 240–249. Springer, 2012.

[110] Diane R Campbell. Predicting plant reproductive success from models of competition for pollination. *Oikos*, pages 257–266, 1986.

[111] Alan Dorin, Tim Taylor, Martin Burd, Julian Garcia, Mani Shrestha, and Adrian G Dyer. Competition and pollen wars: simulations reveal the dynamics of competition mediated through heterospecific pollen transfer by non-flower constant insects. *Theoretical Ecology*, 14(2):207–218, 2021.

[112] Se-Hyung Cho, Ye-Hoon Kim, In-Won Park, and Jong-Hwan Kim. Behavior selection and memory-based learning for artificial creature using two-layered confabulation. In *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 992–997. IEEE, 2007.

[113] Scott L Delp, Frank C Anderson, Allison S Arnold, Peter Loan, Ayman Habib, Chand T John, Eran Guendelman, and Darryl G Thelen. Opensim: open-source software to create and analyze dynamic simulations of movement. *IEEE Transactions on Biomedical Engineering*, 54(11):1940–1950, 2007.

[114] Alireza Askarzadeh. A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm. *Computers & Structures*, 169:1–12, 2016.

[115] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver Press, 2010.

[116] Ajith Abraham, Swagatam Das, and Sandip Roy. Swarm intelligence algorithms for data clustering. In *Soft Computing for Knowledge Discovery and Data mining*, pages 279–313. Springer, 2008.

[117] Swagatam Das, Ajith Abraham, and Amit Konar. Swarm intelligence algorithms in bioinformatics. In *Computational Intelligence in Bioinformatics*, pages 113–147. Springer, 2008.

[118] Ikas Kassabalidis, MA El-Sharkawi, RJ Marks, P Arabshahi, and AA Gray. Swarm intelligence for routing in communication networks. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No. 01CH37270)*, volume 6, pages 3613–3617. IEEE, 2001.

[119] Xin-She Yang and Suash Deb. Cuckoo search: recent advances and applications. *Neural Computing and Applications*, 24(1):169–174, 2014.

[120] LA Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24:1337–1342, 1963.

[121] Anis Farhan Kamaruzaman, Azlan Mohd Zain, Suhaila Mohamed Yusuf, and Amirmudin Udin. Levy flight algorithm for optimization problems-a literature review. In *Applied Mechanics and Materials*, volume 421, pages 496–501. Trans Tech Publ, 2013.

[122] Andy M Reynolds and Mark A Frye. Free-flight odor tracking in drosophila is consistent with an optimal intermittent scale-free search. *PloS One*, 2(4):e354, 2007.

[123] Alexei V Chechkin, Ralf Metzler, Joseph Klafter, Vsevolod Yu Gonchar, et al. Introduction to the theory of lévy flights. *Anomalous Transport*, 129, 2008.

[124] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[125] Carl Anderson. Linking micro-to macro-level behavior in the aggressor-defender-stalker game. *Adaptive Behavior*, 12(3-4):175–185, 2004.

[126] Corrado Gini. *I fattori demografici dell'evoluzione delle nazioni*, volume 8. Fratelli Bocca, 1912.

[127] James Morgan. The anatomy of income distribution. *The Review of Economics and Statistics*, pages 270–283, 1962.

[128] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *ArXiv Preprint ArXiv:1912.06059*, 2019.

[129] Kenneth V Price. Differential evolution. In *Handbook of Optimization*, pages 187–214. Springer, 2013.

[130] Kenneth V Price. Differential evolution: a fast and simple numerical optimizer. In *Proceedings of North American Fuzzy Information Processing*, pages 524–527. IEEE, 1996.

[131] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[132] André Baresel, Harmen Sthamer, and Michael Schmidt. Fitness function design to improve evolutionary structural testing. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 1329–1336, 2002.

[133] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[134] Michio Sugeno. Theory of fuzzy integrals and its applications. *Doct. Thesis, Tokyo Institute of Technology*, 1974.

[135] Didier Dubois, Henri Prade, and Agnès Rico. Residuated variants of sugeno integrals: Towards new weighting schemes for qualitative aggregation methods. *Information Sciences*, 329:765–781, 2016.

[136] Sung-Bae Cho. Fuzzy aggregation of modular neural networks with ordered weighted averaging operators. *International Journal of Approximate Reasoning*, 13(4):359–375, 1995.

[137] Wang Jia and Wang Zhenyuan. Using neural networks to determine sugeno measures by statistics. *Neural Networks*, 10(1):183–195, 1997.

[138] Genichi Taguchi. Introduction to quality engineering: designing quality into products and processes. Technical report, 1986.

Appendix A

Publication List

This is a list of publications during doctoral research in each chapter.

Chapter 1: Introduction

Chapter 2: Traditional Metaheuristic Optimization

Chapter 3:Parametric Study of Metaheuristic

- Willa Ariela Syafruddin, Brahim Benaissa, and Mario Köppen, *"Does the Jaya Algorithm Really Need No Parameters?"*, The 10th International Joint Conference on Computational Intelligence (IJCCI 2018), Seville-Spain, pp. 264-268, Sept. 2018.

Chapter 4: Embedded Metaheuristic for Blockchain Proof-of-Work

- Willa Ariela Syafruddin, Sajjad Dadkhah, and Mario Köppen, *"Blockchain Scheme Based on Evolutionary Proof of Work"*, 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington-New Zealand, pp. 771-776, June 2019.

Chapter 5: Idle Metaheuristic for Flower Pollination Simulation

- Willa Ariela Syafruddin, Rio Mukhtarom Paweroi, and Mario Köppen, *"Behavior Selection Metaheuristic Search Algorithm for the Pollination Optimization: A Simulation Case of Cocoa Flowers"*, Algorithms, Vol.14, pp.230, July. 2021.

Chapter 6: Conclusion and Future Work