COMPARATIVE EVALUATION OF ANT COLONY OPTIMIZATION AND ARTIFICIAL BEE COLONY OPTIMIZATION TO SOLVE TRAVELING SALESMAN PROBLEM (ILORIN-ABEOKUTA-LAGOS-IBADAN-OSOGBO)

BY

HUSSEIN, YUSUF OLAMILEKAN

HND/23/COM/FT/0028

BEING A PROJECTED SUBMITTED TO:

THE DEPARTMENT OF COMPUTER SCIENCE,

INSTITUTE OF INFORMATION, COMMUNICATION AND TECHNOLOGY,
KWARA STATE POLYTECHNIC, ILORIN
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF HIGHER NATIONAL DIPLOMA (HND) IN COMPUTER SCIENCE

SUPERVISED BY:

DR. AGBOOLA, O.M.

JUNE, 2025

CERTIFICATION

This is to certify that this project was carried out by **Hussein, Yusuf Olamilekan** with Matriculation Number **HND/23/COM/FT/0028** as part of the requirements for the award of Higher National Diploma (HND) in Computer Science.

DR. AGBOOLA, O. M.	**DATE**
PROJECT SUPERVISOR	
MR. OYEDEPO, F. S.	**DATE**
HEAD OF DEPARTMENT	
EXTERNAL EXAMINER	**DATE**

DEDICATION

The project is dedicated to the glory of the Almighty God.

ACKNOWLEDGEMENTS

All praise is due to Almighty God, the Lord of the universe. I praise him and thank him for giving me the strength and knowledge to complete my HND program and also for our continued existence on Earth.

I appreciate the utmost effort of my supervisor, DR. AGBOOLA, O.M. whose patience, support, and encouragement have been the driving force behind the success of this research work. He gave useful corrections, constructive criticisms, comments, recommendations, and advice and always ensures that excellent research is done. My sincere gratitude goes to the Head of the department Mr. Oyedepo F.S. and other members of staff of the Department of Computer Science, Kwara State Polytechnic, Ilorin, for their constant cooperation, constructive criticisms, and encouragement throughout the program.

Special gratitude to my parents, who exhibited immeasurable financial, patience, support, prayers, and understanding during the period in which I was busy tirelessly with my studies, special thanks also go to my lovely siblings

My sincere appreciation goes to my friends and classmates.

TABLE OF CONTENTS

ABSTRACT

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization challenge that aims to determine the shortest possible route that visits a set of cities exactly once and returns to the origin. This project presents a comparative evaluation of two nature-inspired metaheuristic algorithms, Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABCO) for solving TSP effectively. ACO simulates the foraging behavior of ants, while ABCO is inspired by the food foraging strategies of honeybee colonies. The research involved adapting both algorithms to TSP, implementing them using Python and Django, and applying them to a dataset of five Nigerian cities (Ilorin, Abeokuta, Lagos, Ibadan, and Osogbo). The performance of each algorithm was assessed based on solution quality, convergence speed, and computational efficiency. Experimental results show that while ABCO achieved a slightly better route with a minimum distance of 590 km, ACO demonstrated faster execution time at 0.18 seconds. Both algorithms successfully satisfied hard constraints like tour completeness and the absence of sub tours, and soft constraints such as minimizing travel distance. This study highlights the practical benefits of swarm intelligence algorithms in route optimization and recommends further exploration of hybrid models for large-scale real-world applications in logistics and transportation.

Keywords: *Traveling Salesman Problem, Ant Colony Optimization, Artificial Bee Colony Optimization, Metaheuristic Algorithms, Route Optimization.*

CHAPTER ONE

1.0 INTRODUCTION

The Traveling Salesman Problem (TSP) is an optimization problem that has caught the interest of researchers for decades. It involves finding the shortest route that visits a set of cities at least once and returns to the origin city, which is the starting point which brings up a problem in fields such as logistics, transportation among others etc. The complexity of Traveling Salesman Problem increases with the number of cities in which has to be reached at least once. As a result, heuristic and metaheuristic algorithms are made use of to provide a near optimal solutions within a reasonable timeframe to the TSP problem. Among multiple metaheuristic algorithms developed to help solve TSP, two significant algorithms to help solve TSP efficiently are Ant Colony Optimization and Artificial Bee Colony Optimization.

ACO is inspired by the foraging behavior of ants, utilizes pheromone trails to guide the search for optimal path. Ant Colony Optimization is a family of nature inspired metaheuristics often applied to finding approximate solution to difficult optimization problems. Despite significantly faster than exact methods, the ACO can still be prohibitively slow, especially if compared to basic problem-specific heuristics. ACO belongs to a growing collection of nature inspired metaheuristics that can be applied to solve various optimization problems (Skinderowicz, 2022). In the context of TSP, artificial ants simulate the behavior of real ants by constructing tours and depositing virtual pheromones to guide the search process. The strength of ACO lies in its positive feedback mechanism and distributed computation, which helps it efficiently explore and exploit the search space.

ABCO mimics the foraging behavior of honeybees, employing a search strategy that enhances the sharing of information among agents to discover solutions. Artificial Bee Colony is one of the groups of "Swarm Intelligence" which refers to the collective behavior of the decentralized and self-organized system, commonly composed of agents that follow uncompleted rules where the communication lead to the evolution of intelligent behaviors (Zuloaga and Moser, 2017). The colony of bees is divided into three types with different method that approaches the food: the employed bees, onlooker bees and scout bees. Employed bees have visited food beforehand and moved the honey bee to the source of the food while onlooker bees are waiting as the area to decide the food source and the scout bees more arbitrarily inside the chosen area (Mridula, Rahmon and Ameer, 2018).

Both ACO and ABC have been successfully applied to various optimization problems, including the TSP. However, despite their individual strengths, there are always ongoing research to determine which algorithm performs better under specific problem instances and parameter settings. A comparative evaluation of these algorithms can provide insights into their effectiveness, efficiency, and robustness in solving the TSP, and help in selecting the most suitable algorithm for different scenarios.

1.1 STATEMENT OF THE PROBLEM

The Traveling Salesman problem (TSP) is a fundamental combinatorial optimization challenge that seeks to determine the shortest possible route for a salesman to visit a set of cities exactly once and return to the starting point and as the number of cities increases, the complexity of the problem escalates making it hard to find optimal solution. Using these algorithms ACO & ABCO, the quality of solution would be enhanced when applied to various instances of TSP, it should make larger size of TSP to be computational feasible to be able to find an optimal solution.

1.2 AIM AND OBJECTIVES OF THE STUDY

The aim of this project is to solve the TSP using Ant Colony Optimization and Artificial Bee Colony Optimization and also to compare the solution of the implemented algorithms based on distance, time and memory usage.

The objectives are as follows:

i. adaptation of the Traveling Salesman Problem to the algorithms (ACO & ABCO);
ii. implementation of the Adapted Algorithms; and
iii. evaluation of ACO and ABCO for solving TSP in term of time and memory usage.

1.3 SIGNIFICANCE OF THE STUDY

The significance of the study is the comparative analysis of both algorithms (ACO & ABCO) in the field of optimization, precisely to solve the traveling salesman problems. By comparing algorithms, TSP can be solved by evaluating the performance of ACO and ABCO based on solution quality, convergence speed and its ability to satisfy the constraints and then figuring out the best algorithm to solve TSP considering the criteria of the constraints.

The findings of this research can also help in industries that rely on efficient routing solution such as logistics, transportation and supply chain management. Businesses can make decisions about which optimization technique to implement, leading to improved operational efficiency.

1.4 SCOPE OF THE STUDY

This study focuses of the comparison evaluation of Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABCO) as approaches to solving the Travelling Salesman Problem (TSP) using the dataset of five cities which are (Ilorin-Abeokuta-Lagos-Ibadan-Osogbo).This research will focus on the formulation of the TSP, where the objectives is to find the shortest possible route that visits a set of cities (Ilorin-Abeokuta-Lagos-Ibadan-Osogbo) exactly once and return to the starting city i.e. Ilorin.

1.5 ORGANIZATION OF THE STUDY

This project write up is organized into five chapters. Chapter one covers general introduction, aim and objectives, statement of the problem, significance of the study, scope of the study, organization of the study and definition of terms. Chapter two covers literature review, which contains review of past works, concept of ACO & ABCO. Advantages and disadvantages of ACO and ABCO and overview of ACO and ABCO to solve TSP. Chapter three explains the project methodology, analysis of the existing problem, gaps in the current research, description of the proposed solution. Chapter four explains the results and analysis of the research. Chapter five presents the summary of the findings, re commendations and conclusion.

1.6 DEFINITION OF TERMS

1. **Traveling Salesman Problem (TSP):** a combinatorial optimization problem that seeks to find the shortest possible route that visits a set of cities exactly once and returns to the starting city.

2. **Ant Colony Optimization (ACO):** a bio-inspired optimization algorithm based on the foraging behavior of ants. It uses a technique to find optimal paths by simulating how ants deposit pheromones on paths they traverse.
3. **Artificial Bee Colony Optimization (ABCO):** a metaheuristic algorithm inspired by the foraging behavior of honeybees. It involves a population of analytical bees that explores the solution space, sharing information of food sources and their quality.
4. **Heuristic Algorithm:** a problem-solving approach that employs particular methods that may not be optimal but are sufficient for reaching on immediate goal.
5. **Metaheuristic Algorithm:** a higher-level procedure designed to guide other heuristic to explore the solution space more is effectively.
6. **Pheromone Trail:** in ACO, a virtual trail laid down by ants that represents the desirability of a path.
7. **Food Source:** in ABCO, a potential solution to the optimization problem, singular to a food source for bees.
8. **Benchmark Instances:** Standardized problem instances used to evaluate the performance of optimization algorithms.
9. **Convergence Speed:** the rate of which an optimization algorithm approaches a solution.
10. **Solution Quality:** a measure of how good a solution is in terms of the objective function. It is assessed by the total distance produced by the algorithm with shorter distances indicating better solutions.

CHAPTER TWO

2.0 LITERATURE REVIEW

Liu, Lee, Wenshi, and Guo (2022) worked on DAACO: Adaptive Dynamic Quantity of Ant ACO Algorithm to Solve the Traveling Salesman Problem. The paper aims to improve the Ant Colony Optimization (ACO) algorithm for solving the Traveling Salesman Problem (TSP). The main goal is to create a version of ACO that: Adapts the number of ants based on the size of the problem, avoids getting stuck in poor solutions and finds better paths more quickly. The authors introduced a new ACO algorithm called DAACO. It uses a convex hull and K-means clustering to decide how many ants to use, it also introduces a new local search strategy to help ants choose better paths faster and combines this with the 3-opt algorithm to improve the final solution. The problems of the study were; traditional ACO methods often take too long to find good solutions, get stuck in local optimum (not the best solution) and use a fixed number of ants that may not suit all problem sizes. The research discussed the DAACO algorithm: Performs better than older ACO versions (like DFACO and DEACO), achieves faster results with less error, finds better paths in large TSP problems and is more stable and adaptable across different types of problems. The study conclude that, the DAACO algorithm is a strong improvement over older ACO methods. It adjusts the number of ants and uses smart strategies to avoid bad solutions. It works well on many TSP problems and can be useful for solving other optimization problems too. Future work should explore using DAACO for asymmetric TSP problems (where the distance from A to B is not the same as B to A). The authors also suggest applying DAACO to other complex optimization tasks.

Meena and Ramadass (2020) worked on Improving Ant Colony Optimization Efficiency for Solving Optimization Problems. The aim was to improve the efficiency of the Ant Colony Optimization (ACO) algorithm and to make the algorithm faster and better at finding the best solutions for difficult problems like the Traveling Salesman Problem (TSP). In the methodology, the researchers changed the way ants choose paths by introducing a better way of scoring routes. They used a new update rule to help ants find better paths faster and they tested their improved method on several TSP problems and compared it with other existing algorithms. The research gaps discussed Traditional ACO algorithms sometimes take too long to find the best route or get stuck in bad solutions. There was a need for an improved algorithm that solves problems faster and produces better results. The findings show that; the new improved ACO showed faster performance and found better solutions than older methods, it took less time to reach the best solution and avoided getting stuck in bad options. The results proved that the improved algorithm is more effective in solving the TSP. The study concluded that the improved ACO works well for solving complex optimization problems like TSP. It is faster and gives better solutions, making it a useful tool for such problems. The authors suggest further testing on more varied problems. They recommended adjusting the new rules to make the algorithm even better.

Bin Shahadat, Akhand, and Kamal (2022) worked on Visibility Adaptation in Ant Colony Optimization for Solving Traveling Salesman Problem. The goal of this study is to improve the Ant Colony Optimization (ACO) algorithm by adjusting how ants perceive the visibility of paths. The objective is to make the algorithm find better solutions faster by dynamically changing visibility based on the problem’s needs. The researchers introduced a new method where the visibility value, which influences ants' choices, adapts during the search process. They used mathematical models to update this visibility based on how good the routes are. The method was tested on standard problems like the Traveling Salesman Problem (TSP), and results were compared with traditional ACO. The gaps of study mentioned include; Traditional ACO algorithms use fixed visibility values, which might not work well for different parts of a problem. Sometimes, this can slow down the search or lead to suboptimal solutions. There was a need to make the visibility more flexible and adaptive for better results. In the findings, the new adaptive visibility method improved the algorithm's performance. It found shorter routes more quickly and avoided getting stuck in bad solutions. The results showed that adjusting visibility dynamically helps the algorithm explore and exploit better, leading to higher accuracy and efficiency in solving TSP and similar problems. The study concluded that adaptive visibility in ACO makes the algorithm more effective. It can find better solutions faster and is more reliable compared to traditional fixed-visibility approaches. The method is promising for solving complex optimization problems. The researchers recommend further testing this approach on larger and more varied problems. They also suggest exploring other ways to adapt parameters during the search process, which could improve the algorithm even more.

Rufai, Usman, Olusanya, and Adedeji (2021) presented a research worked “Solving Travelling Salesman Problem Using an Improved Ant Colony Optimization Algorithm”. The main goal of the research is to develop a better version of the Ant Colony Optimization (ACO) algorithm that can solve the Traveling Salesman Problem (TSP) faster and more accurately. The focus is on finding the shortest route visiting 52 cities in Nigeria, starting from Abuja. The researchers used the improved ACO algorithm, which mimics how real ants find food. They adjusted parameters like pheromone levels, exploration, and exploitation to make the algorithm work better. They tested it on data about 52 Nigerian cities, using MATLAB software. The algorithm builds routes, updates pheromones, and finds the shortest path through the cities. Traditional methods like linear programming and older algorithms struggle with large problems like TSP because they take a lot of time or don't find the best route. Older algorithms often get stuck in bad solutions or are slow. There was a need for an improved method that could solve large TSPs quickly and accurately. The study finds out that the improved ACO found very good routes, with the shortest path length of 5866.92 units. It showed faster convergence and avoided getting stuck in bad solutions better than traditional ACO. Different parameter settings affected the results, but overall, the improved algorithm was very effective. It achieved about 98% accuracy compared to traditional methods and found shorter routes for the Nigerian cities. The study concluded that the improved ACO algorithm is efficient and can find the best or near-best routes in a short time. It works well for solving large TSPs, like the 52 cities in Nigeria. The algorithm can be used in real-world routing problems, saving time and effort. The researchers suggest trying the algorithm on larger datasets and comparing it with other advanced algorithms. They also recommend fine-tuning parameters further to improve accuracy and speed.

Xing and Tu (2020) worked on A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem. The main aim is to solve the Traveling Salesman Problem (TSP) more effectively by combining a Graph Neural Network (GNN) with Monte Carlo Tree Search (MCTS). The goal is to make more accurate and reliable decisions when finding the shortest tour that visits each city exactly once. In the Methodology, the authors developed a method using Graph Neural Networks to learn patterns in graphs. This network gives a probability for each vertex being chosen next in a tour, then, they used Monte Carlo Tree Search (MCTS) to explore better options by simulating many possible paths. They trained and tested the model using generated datasets with different graph sizes (e.g., TSP20, TSP50, TSP100) and they compared their results to traditional algorithms and other learning-based methods. Problem mentioned of the researcher were; existing learning-based methods often make unreliable decisions because they only have one chance to choose the best path and cannot go back, traditional methods depend on manually created rules, which are hard to design and there's a need for methods that work well on both small and large graphs. During the findings**,** the proposed GNN-MCTS method performed better than other recent learning-based methods, especially in finding shorter tours and the method showed good generalization and adaptability. The study concludes that combining Graph Neural Networks with Monte Carlo Tree Search is a powerful way to solve the TSP. It performs better than other similar methods and can be used for larger problems, showing potential for solving other complex optimization tasks. The authors suggest this method can be extended to solve other NP-hard problems. They also recommend improving implementation speed by rewriting the algorithm in faster languages like C++, and further exploring better designs for value functions.

Prates, Avelar, Lemos, Lamb, Vardi (2019) worked on Learning to Solve NP-Complete Problems: A Graph Neural Network for Decision TSP. The aim is to see if Graph Neural Networks (GNNs) can learn to solve a complex problem called the Traveling Salesperson Problem (TSP), specifically its decision form, with very little help and to investigate whether

GNNs can be trained to decide if a route with a certain cost exists in a graph and how well they can predict the route's cost. In the methodology, they created a GNN model that assigns special data representations (embedding) to parts of a graph (vertices and edges), the model uses several rounds of message passing, where information flows between connected nodes and updates these representations, the model was trained with examples where they knew the optimal route cost and created slightly easier or harder problems (target costs slightly smaller or larger than the true cost) and the training was done using a technique called stochastic gradient descent, with many graphs of 20-40 nodes. The gaps identified by the researcher shows that; it’s unclear if neural networks can learn to solve such problems, especially when they involve numeric data like weights, not just symbolic relationships and previous methods mostly focused on smaller problems or simpler problems like SAT (Boolean satisfiability). The findings shows that the GNN model was able to correctly decide if a route exists with over 80% accuracy on new, unseen graphs and the results show GNNs are powerful enough to handle complex problems involving both symbols and numbers. The study conclude that, GNNs can successfully learn to approximate solutions to NP-Complete problems like TSP with very little supervision and the model's predictions get better as they deviate more from the exact cost, and it can even work on larger graphs than it was trained on. Future work should include applying this approach to real-world data and exploring how well it scales.

Pan, Jin, Ding, Feng, Zhao, Song, Bian (2023) worked on H-TSP: Hierarchically Solving the Large-Scale Travelling Salesman Problem. The aim is to create a new method (H-TSP) that solves very large Traveling Salesman Problems (TSPs) efficiently. The goal is to reduce the time it takes to find good solutions without losing much accuracy, even for problems with up to 10,000 cities. The authors used a hierarchical deep reinforcement learning approach. Their method splits a large problem into smaller parts using two models: An upper-level model that picks parts of the problem to solve & a lower-level model that solves those smaller parts. These two models work together and are trained using reinforcement learning to solve the TSP in a "divide and conquer" way. The gaps of study mentioned existing learning-based methods either take too long to compute or can't handle large problems well. Most models work only for small-scale TSPs. The authors wanted to solve large problems quickly and accurately, something that other methods struggle with. In the discussion of the study, H-TSP gives results similar to the best search-based methods but is much faster—up to 100 times quicker, it performs well across various TSP sizes (from 1,000 to 10,000 nodes), the study also showed that their method generalizes well to even larger problems (like 50,000 nodes) and replacing the lower-level model with a traditional method (LKH-3) gives better quality but increases time. The study conclude H-TSP is a fast and effective way to solve large TSPs. It balances between speed and solution quality. The method can also be adapted to other problems, like vehicle routing or scheduling tasks. The authors suggest improving the solution quality by using better solvers (like LKH-3) at the lower level. They also recommend applying this hierarchical method to other large optimization problems in the future.

Dahan, Hindi, Mathkour, AlSalman (2019) worked on Dynamic Flying Ant Colony Optimization (DFACO) for Solving the Traveling Salesman Problem. Aim and objective of the study was to develop and improve a new version of the ant colony algorithm called DFACO, which helps solve the Traveling Salesman Problem (TSP) faster and better and to make the algorithm more efficient by avoiding getting stuck in bad solutions and reducing solving time.

Research methodology used were; the researchers modified the existing Flying Ant Colony Optimization (FACO) algorithm. They allowed ants to choose more flexible neighbors and decide dynamically how many neighbors to consider, they added rules to balance exploring new routes and using the best-known routes. The 3-Opt algorithm was embedded to make solutions better by fixing problems called "local minima, some ants were made to "fly" (use the new method), and others "walk" (use traditional method) and they tested the new method on many sets of TSP problems and compared it with other algorithms. The gaps mentioned include; many existing algorithms take too long or get stuck in bad solutions when trying to solve TSP, previous methods didn't balance exploring new routes and using known good routes well enough and there was a need to create an algorithm that finds better solutions faster and avoids dead-ends. The study concluded that the improved DFACO algorithm works well for solving the Traveling Salesman Problem and it finds high-quality solutions quickly and avoids common problems like getting stuck in bad solutions. The authors suggest further testing on different datasets to improve the algorithm. Implementing this method in real-world problems could help in logistics, routing, and other areas requiring efficient route planning.

Dhanasekar, Dash, and Uthaman (2022) conducted a research titled "A Branch and Bound Algorithm to Solve Travelling Salesman Problem (TSP) with Uncertain Parameters." The aim is to develop a new algorithm that can solve the Traveling Salesman Problem (TSP) even when the data is uncertain or fuzzy. The goal is to find the best route that visits all cities once with the least cost, considering uncertain parameters modeled as fuzzy numbers. The researchers used a branch and bound approach that involves several steps: Firstly, they reduce the cost matrix by subtracting the smallest values in each row and column to simplify the problem. Then, they calculate penalties for not choosing certain routes, based on fuzzy costs. They divide possible routes into groups, evaluate their costs, and eliminate routes that are less promising and they repeat this process until the best possible route is found that meets all the conditions, ensuring no sub-tours are included. The problem of the study was; Traditional methods struggle with uncertain data because they use crisp (exact) numbers, which don't reflect the real vagueness or hesitation in data. Existing solutions often don't satisfy route conditions properly and may give less accurate results when data is fuzzy. The findings show that the proposed algorithm effectively finds the optimal route considering uncertain data, modeled as intuitionistic fuzzy numbers. Tests show that it successfully finds routes that satisfy all conditions, and the results are valid and reliable. The method is also more suited for real-world problems where data uncertainty is common. The study concluded that the developed branch and bound method is efficient and easy to understand. It can find the best route for TSP even when data is fuzzy and uncertain. The results confirm that this approach works well and can be used for solving similar complex problems. The researchers suggest further testing with different data sets to improve accuracy. They also recommend applying the method to real-world problems like logistics and transportation, where data uncertainty often exists. Fine-tuning the algorithm could make it even more effective.

Kiran and Beskirli (2024) worked on A New Approach Based on Collective Intelligence to Solve Traveling Salesman Problems. The aim is to develop a new algorithm that avoids stagnation and produces high-quality solutions for TSP. It leverages collective intelligence using footprints and neighborhood optimization. The objective is to create a method that constructs solutions based on collective footprints, improves solutions iteratively, and maintains diversity to prevent premature convergence. The proposed method involves two phases: path construction and path improvement. Path Construction Phase: Artificial agents called "constructors" start at city nodes and select the next city based on a probability that considers the number of footprints (collective intelligence) and the distance between nodes. Footprints are left on visited arcs, and more footprints increase the chance of selecting that arc. No pheromone evaporation mechanism is used. Path Improvement Phase: The best constructed solution is refined using neighborhood operators such as random insertion, subsequence insertion, and reversal. Multiple agents, called "improvers," apply these operators to improve the solution iteratively. The algorithm repeats these phases until a stopping criterion is reached, aiming to find the shortest possible route. The gaps addressed was; Standard swarm algorithms like Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), and Particle Swarm Optimization (PSO) tend to stagnate or converge prematurely due to pheromone mechanisms or random initial solutions. Existing algorithms also lack effective diversity maintenance, potentially leading to sub-optimal solutions and stagnation in large TSP instances. In the discussion of findings, the comparison with other algorithms like ABC, ACO, hierarchical algorithms, and hybrid techniques demonstrates that the proposed method performs well and often yields better solutions, especially in larger and more complex TSP instances. Future research should test this approach on different types of combinatorial and discrete optimization problems. Further parameter tuning and hybridization with other algorithms may improve its performance. Extending the algorithm to dynamic or real-time TSP applications could be a valuable direction.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 THE METHODOLOGY

- Ant Colony Optimization Algorithm
- Artificial Bee Colony Optimization Algorithm

3.1.1 Ant Colony Optimization Algorithm

Ant Colony Optimization (ACO) Algorithm is a metaheuristic algorithm inspired by the foraging behavior of real ants. It is used to solve combinatorial optimization problems by simulating how ants find the shortest path between their colony and a food source using pheromones. In ACO, a set of artificial ants explores potential solutions, deposits pheromones, and gradually converge towards an optimal or near-optimal solution. It was first introduced by Marco Dorigo in 1992.

ACO & Adapted ACO Algorithm for TSP

- Define the fitness function: Establish a fitness function to assess the quality of each solution. This function should aim to minimize the number of penalties violated which ensures the smoothness of the tour.
- The parameter setting: Determine the parameters of the ACO based on the characteristics of the problem. Key parameters include: Number of ants, number of iterations, evaporation rate, pheromone influence, and the stopping criterion.

- Generate solutions: Create initial tours from a city through the rest of cities through randomization. These solutions should adhere to the specified constraints to avoid penalty.
- Evaluate the solutions: Calculate the fitness of each solution using the defined fitness function.
- Update pheromones: Update the pheromones by selecting the best tour from the solutions through the use of the parameters that has been set.
- Repeat the iterations: Continue this process until a stopping criterion is met, such as the maximum number of iterations is met or convergence is achieved.
- Select the best tour: Choose the best and shortest tour to represent the solution.

3.1.2 Artificial Bee Colony Optimization Algorithm

Artificial Bee Colony Optimization (ABCO) Algorithm is a swarm intelligence-based optimization algorithm inspired by the foraging behavior of honeybee colonies. It is used to solve complex optimization problems by mimicking how bees search for food sources, share information, and optimize their choices. It was first proposed by Karaboga in 2005.

ABCO and Adapted ABCO Algorithm for TSP

- Define the fitness function: Establish a fitness function to assess the quality of each solution. This function should aim to minimize the number of penalties violated which ensures the smoothness of the tour.

- The parameter setting: Determine the parameters of the ABCO based on the characteristics of the problem. Key parameters include: Colony size, maximum of iterations, lower bound, upper bound, and the stopping criterion.
- Generate each phase solutions: Create solutions from all the phases i.e. Employed bee phase, onlooker bee phase and scout bee phase by each checking the solution and fitness before moving to the next phase. These solutions should adhere to the constraints.
- Evaluate the solutions: Calculate the fitness of each solution using the defined fitness function.
- Update the solutions: Update the solution by selecting the best fitness value.
- Repeat the iterations: Continue this process until a stopping criterion is met, such as the maximum number of iterations is met or convergence is achieved.
- Select the best tour: Choose the best and shortest tour to represent the solution.

3.2 ANALYZING THE PROBLEMS OF EXISTING SYSTEM

The main problem with the Traveling Salesman Problem (TSP) is its NP-hard complexity, meaning that finding the optimal solution becomes computationally intractable as the number of cities (or nodes) increases, requiring exponential time and resources. This problem of traveling salesman problem is improved by introducing the existence of a set of constraints, which is typically divided into two types: Hard and Soft constraints.

Hard constraints are requirements that must be satisfied for a solution to be considered valid. Violating these constraints typically results in the solution being discarded or heavily penalized, while **Soft** constraints are desirable but not strictly required. Violating these constraints results in a penalty that affects the quality (fitness) of the solution but does not render it invalid. The following outlines the hard and soft constraints formulated through our research and understanding of the traveling salesman problem in a visited city.

Hard Constraints

1. Completeness of the Tour:

- The solution must visit every city exactly once and return to the starting city.

2. Feasibility of the Path:

- The path must be a valid cycle (Hamiltonian cycle) without any disconnections.

3. No Sub tours**:**

- The solution must not contain any sub tours (smaller cycles that do not include all cities).

Soft Constraints

1. Minimization of Tour Length:

- The primary objective of TSP is to minimize the total distance of the tour.

2. Balanced Exploration and Exploitation:

- The algorithm should balance exploration (searching new areas of the solution space) and exploitation (refining known good solutions).

3. Computational Efficiency:

- The algorithm should converge to a good solution within a reasonable time frame.

4. Robustness:

- The algorithm should perform consistently across different instances of TSP (e.g., different numbers of cities or city distributions).

Penalty for Soft Constraints

Soft constraints should have a smaller penalty to encourage optimal solutions without completely discarding less ideal ones.

- The fitness function directly incorporates the total tour length. Longer tours receive higher penalties.
- Poor balance may lead to suboptimal solutions or slow convergence, which can be measured by the algorithm's performance over iterations.
- Slower convergence or higher computational cost can be penalized by comparing the runtime or number of iterations required to reach a solution.
- Inconsistent performance can be penalized by evaluating the variance in solution quality across multiple runs or problem instances.

3.3 STEP-BY-STEP APPROACH TO THE ACO ALGORITHM

1. Initialize Parameters

- Set parameters such as:
    - Number of ants (m)

    - Number of iterations (T)
    - Evaporation rate (p)
    - Pheromone influence (α)
    - Heuristic information influence (β)
- Initialize pheromone levels (T) on all paths with a small positive value.

2. Construct Solutions

- Each ant starts at a randomly chosen node (or city in TSP.
- Ants construct solutions iteratively by choosing the next node probabilistically based on:

$$P_{ij} = \frac{(Tij)\alpha(\eta ij)\beta}{\sum_{k\in Ni}(Tik)\alpha(\eta ik)\beta}$$

where:

P_{ij} = Probability of selecting node j from node i

T_{ij} = Pheromone level on edge (i,j)

η_{ij} = Heuristic value (e.g., $1/d_{ij}$ for TSP, where d_{ij} is the distance)

α, β = Control the influence of pheromone vs. heuristic information

N_i = Set of feasible next nodes

3. Evaluate Solutions

- Compute the quality of each ant's solution (e.g., total path length for TSP).
- Identify the best solution in the current iteration.

4. Update Pheromone Trails

- Evaporate pheromone on all paths to prevent stagnation:

$T_{ij} = (1 - p)T_{ij}$

- Deposit pheromone based on solution quality:

$$T_{ij} = T_{ij} + \sum_{k=1}^{m} \Delta T_{ij}^{k}$$

where:

p = Evaporation rate ($0 < p < 1$)

$\Delta T_{ij}^{k} = \frac{Q}{L^k}$ if ant k used edge (i,j), otherwise 0

Q = Constant pheromone amount

L^k = Total cost (or path length) of ant k's solution

5. Check Termination Criteria

- If the maximum number of iterations or a convergence criterion is met (e.g., no improvement over a certain number of iterations), stop the algorithm.
- Otherwise, go back to Step 2 for the next iteration.

3.3.1 Adaptation of Ant Colony Optimization Algorithm to Traveling Salesman Problem

We have modified both ACO and ABCO to effectively solve TSP by representing certain cities (i.e. our dataset), distances and paths in a way that aligns with their optimization processes to obtain the best output.

The Steps is as Follows:

Step 1: Initialize the data and parameters

i. State the input data used for the optimization problems including the locations that the salesman must visit which are llorin, Lagos, Abeokuta, Ibadan, Osogbo) and must return to the starting point which is Ilorin.

ii. Set the parameters such as (Number of ants, Number of iterations, Evaporation rate, Pheromone influence, Heuristic information influence (Alpha, Beta and Q).

Population size = 05

Max iterations=10

Pheromone Evaporation Rate = 0.5

Pheromone influence = 1.35135

Alpha = 1

Beta = 1

Q=1000

i. Define the problem as a graph which represents solution components i.e (the distance of the cities which ACO want to work on).

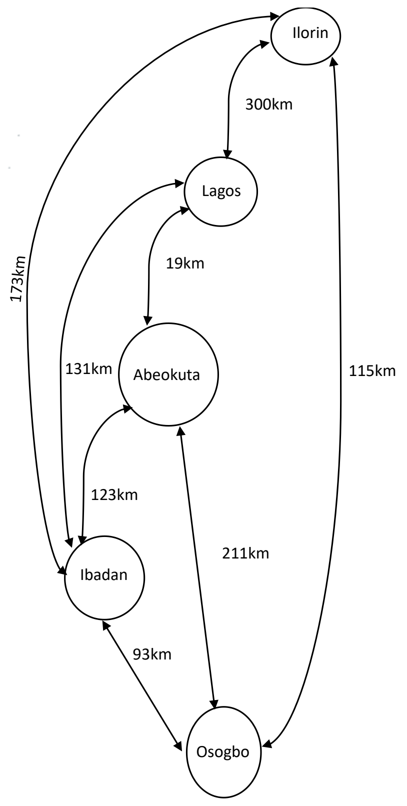


Fig 3.3.1: Block diagram of the tour distance covered

Step 2: Construct Solution:

i. Select Random City

ii. Build Tour from Current City to Unvisited Cities

iii. Return Tour to the City in which the Salesman Started

Step 3: Evaluate Solution:

i. Calculate Distance between each city which is already indicated in Step 1

ii. Any City is randomly Selected by Ant

iii. Build Tour from Current City to Unvisited Cities using Stochastic mechanism i.e.

$$P_{ij} = \frac{(Tij)\alpha(\eta ij)\beta}{\sum_{k\in Ni}(Tik)\alpha(\eta ik)\beta}$$

where

T_{ij} = Pheromone level on edge

$n_{ij} = \frac{1}{dij}$

α = Constant 1

β = Constant 1

ij = Set of unvisited cities

ii. Compare the random value condition to the solution set i.e (Random value <= solution set) with the cities and choose the next selected cities.

iii. More Ant from the randomly selected city to the next selected city

iv. Repeat process to select the next 3 cities starting from the selected cities.

v. When there is no unvisited city left, Move Ant from current city to initial city.

Hard Constraints

- The solution must visit every city exactly once and return to the starting city.
- The path must be a valid cycle (Hamiltonian cycle) without any disconnections.
- The solution must not contain any sub tours (smaller cycles that do not include all cities).

Soft Constraints

- The primary objective of TSP is to minimize the total distance of the tour.
- The algorithm should balance exploration (searching new areas of the solution space) and exploitation (refining known good solutions).
- The algorithm should converge to a good solution within a reasonable time frame.
- The algorithm should perform consistently across different instances of TSP (e.g., different numbers of cities or city distributions).

vi. To satisfy the soft constraints, repeat the process for multiple iterations by selecting a random visit to other cities and after returning to the initial city, determine the best tour that has the less time to complete the tour.

Step 4: Update Pheromones

i. Check for the best and shortest tour to receive more pheromones to make it more attractive for the next iterations over the other tours.

Step 5: Check Termination Criterion

i. Repeat Step 3 and 4 until the termination criterion is met.
ii. If the maximum number of iterations is met, stop the algorithm. Otherwise, go back to step 3 for the next iteration.

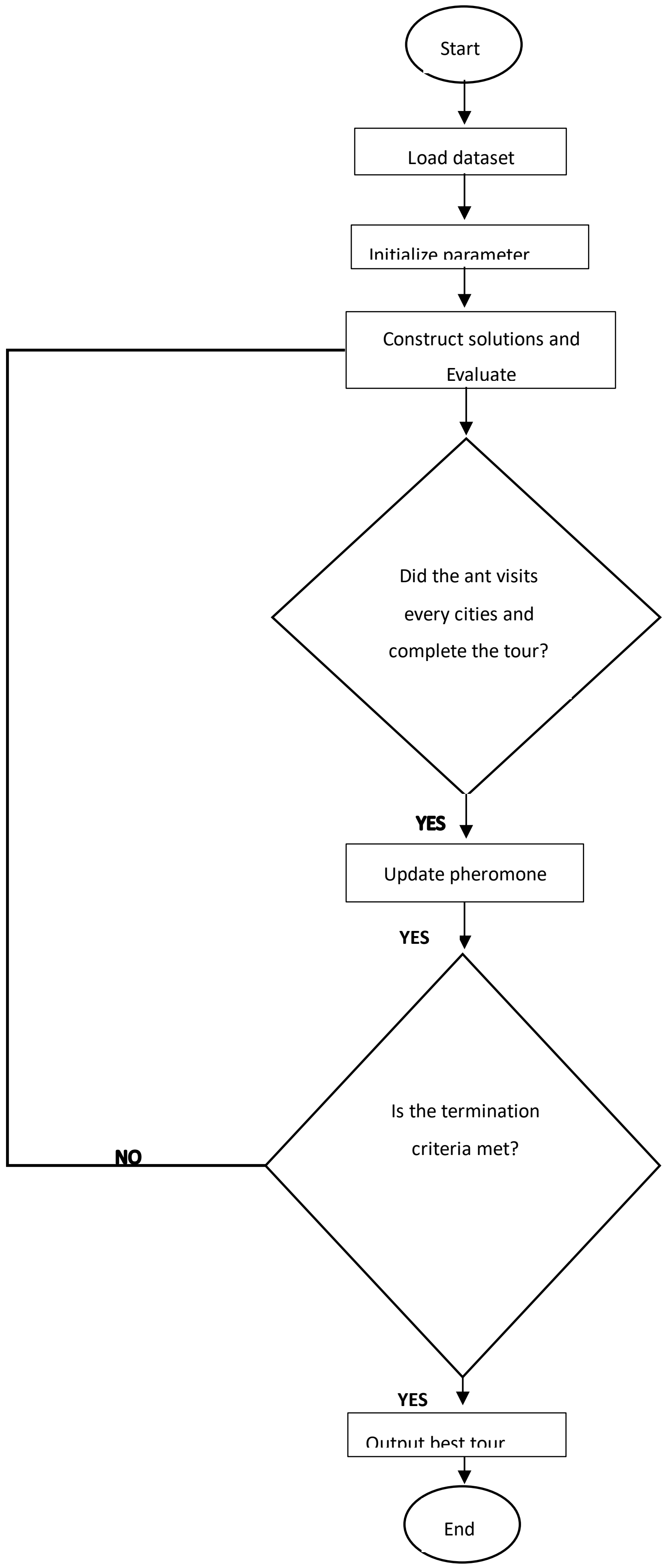

Start
Load dataset
Initialize parameter
Construct solutions and Evaluate
Did the ant visits every cities and complete the tour?
YES
Update pheromone
YES
Is the termination criteria met?
NO
YES
Output best tour
End

Fig 3.3.2: Ant Colony Optimization Flowchart

By applying the Ant Colony Optimization Algorithm to this Traveling Salesman Problem, we can systematically construct solutions that visits all cities once and return to the starting city, and satisfy both hard and soft constraints.

3.4 Step-By-Step Approach of ABCO Algorithm

1. Initialize the Population

- Define the number of food sources N (equal to the number of employed bees).
- Define control parameters:
  - Colony size (number of bees),
  - Maximum number of iterations,
  - Limit parameter (used to abandon poor solutions).
- Randomly initialize the food source positions X_i within the search space:

  $X_{i,j} = X_{\min,j} + \text{rand}(0,1).(X_{\max,j} - X_{\min}$, where $X_{\min,j}$ and $X_{\max,j}$ are the lower and upper bounds of the j-th dimension.

2. Evaluate the Fitness of Each Food Source

- Compute the fitness $f(X_i)$ of each food source using the objective function.

3. Employed Bee Phase (Exploration & Exploitation)

- Each employed bee selects a neighboring food source and generates a new solution:

  $V_{i,j} = X_{i,j} + \phi_{i,j} .(X_{i,j} - X_{k,j})$ where:
  - $V_{i,j}$ is the new candidate solution.
  - k is a randomly chosen food source index $(k \neq i)$

- $\phi_{i,j}$ is a random number in [-1, 1] to control perturbation.

- Apply a greedy selection: If V_i is better than X_i, replace X_i with V_i.

4. Onlooker Bee Phase (Exploitation - Selecting Good Food Sources)

- Compute the probability P_i of each food source being selected:

$$P_i = \frac{f(Xi)}{\sum_{j=1}^{N} f(Xj)}$$

- Each onlooker bee selects a food source based on P_i and applies the same neighborhood search as employed bees.

5. Scout Bee Phase (Abandonment & Exploration of New Areas)

- If a food source is not improved for a certain number of iterations (defined by limit), it is abandoned.
- The scout bee replaces it with a new randomly generated solution.

6. Repeat until Stopping Criterion is met

- Steps 2-5 are repeated until a termination condition is met (e.g., max iterations or convergence).

3.4.1 Adaptation of Artificial Bee Colony Optimization Algorithm to Traveling Salesman Problem

Step 1: Initialize the initial population of routes randomly

i. Set the parameters such as of colony size, maximum number of iterations, lower bound, and upper bound.

Colony Size = 05,

Maximum numbs of iteration = 100

Lower Bound = -10

Upper Bound = 10

ii. Compute the fitness value of each of each food source using the objective function i.e

$D^i = \sum \alpha(C_i, C_i+1) + \alpha(C_N,C1)$

$F^i = \frac{1}{1+Di}$

Step 2: Employed Bee phase:

i. Each employed bee modifies its current route to find a better one.

ii. To modifier, employed bees needs to swap two cities in the route to create a new candidate solution.

iii. Calculate the New Route's Fitness using the objective function as stated in StepI

iv. Apply Greedy Solution i.e if the new route has a better (shorter) distance, replace the old route. Otherwise, the bee keeps the original route.

Hard Constraints

- The solution must visit every city exactly once and return to the starting city.
- The path must be a valid cycle (Hamiltonian cycle) without any disconnections.
- The solution must not contain any sub tours (smaller cycles that do not include all cities).

Soft Constraints

- The primary objective of TSP is to minimize the total distance of the tour.
- The algorithm should balance exploration (searching new areas of the solution space) and exploitation (refining known good solutions).
- The algorithm should converge to a good solution within a reasonable time frame.
- The algorithm should perform consistently across different instances of TSP (e.g., different numbers of cities or city distributions).

Step 3: Onlooker Bee phase:

i. Each onlooker bee selects a TSP route based on fitness i.e the higher the fitness value, the higher the probability to be selected.

ii. Onlookers modify selected routes by either swap, inverse or shift modifications.

iii. Evaluate new route and update if new fitness is better.

Step 4: Scout Bee Phase:

i. If a route does not improve after a certain number of iterations, it is abandoned.

A scout bee replaces it with a new random generated route.

Step 5: Check Termination Criteria

i. Repeat step 2-5 are repeated until the termination condition are met (maximum number of iterations is reached and convergence).

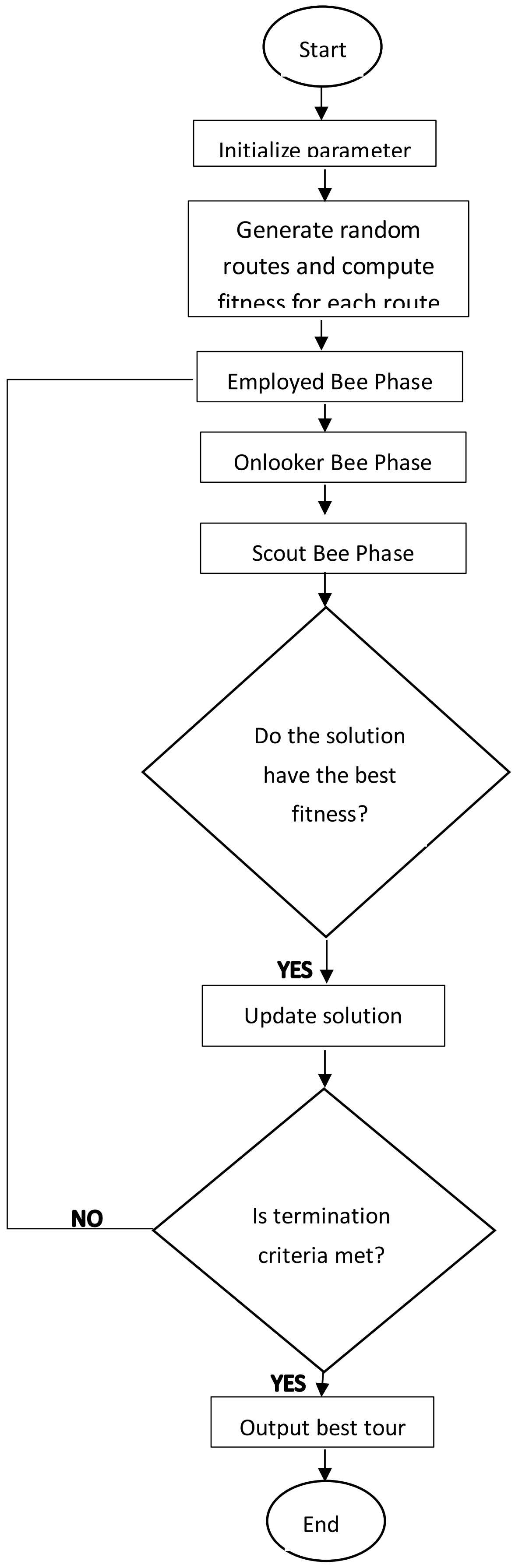

Start
Initialize parameter
Generate random routes and compute fitness for each route
Employed Bee Phase
Onlooker Bee Phase
Scout Bee Phase
Do the solution have the best fitness?
YES
Update solution
Is termination criteria met?
NO
YES
Output best tour
End

Fig 3.4.1: Artificial Bee Colony Optimization Flowchart

3.5 ANALYSIS OF THE PROPOSED SYSTEM

This section entails step by step solutions of the existing problem of traveling salesman problem by using the proposed algorithms; the experiment will be carried out by using dataset of a traveling salesman obtain from five different cities e.g., Ilorin-Lagos-Abeokuta-Ibadan-Osogbo.

3.6 THE ADVANTAGES OF THE PROPOSED SYSTEM

The proposed system offers several key advantages:

- Enhanced Optimization Accuracy: By utilizing nature-inspired algorithms like Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABCO), the system effectively searches for near-optimal solutions to the Traveling Salesman Problem (TSP), which is NP-hard and difficult to solve with traditional methods.
- Constraint Handling: The system incorporates hard constraints (such as visiting each city exactly once, valid Hamiltonian cycles, and avoiding sub tours) and soft constraints (like minimizing total tour length and balancing exploration and exploitation), ensuring feasible and efficient solutions.
- Adaptability and Flexibility: Both algorithms are adapted specifically for TSP, allowing them to handle dataset variations (such as different city configurations) robustly, ensuring consistent performance across different problem instances.
- Balance of Exploration and Exploitation: The algorithms are designed to explore new solutions while refining promising ones, reducing chances of getting trapped in local optima.

- Potential for Improved Cost and Time Savings: By finding shorter and more optimal routes, the system can significantly reduce travel costs and time, providing practical benefits in logistics and route planning.

CHAPTER FOUR

IMPLEMENTATION, RESULT AND DISCUSSION

4.1 INTRODUCTION

This chapter provides an overview of the implementation details, presents the results obtained, and discusses the findings of the traveling salesman problem by comparative evaluation of Ant Colony Optimization (ACO) algorithm and Artificial Bee Colony Optimization (ABCO) algorithm in python. The system aims at evaluating and implementing ant colony optimization algorithm and artificial bee colony optimization algorithm to solve traveling salesman problem. These algorithms are used to solve the traveling salesman problem effectively.

4.2 SYSTEM IMPLEMENTATION

The Implementation of the Traveling Salesman problem was carried out using Ant Colony Optimization, Artificial Bee Colony Optimization algorithm and Python within the Django framework, where data is managed through the route animation. The performance of the two algorithm is evaluated and compared based on their ability to generate the best tour with the shortest time. It involves the steps below:

1. **Data Preparation**

- The system retrieve route data from the cities stored in the database (Ilorin - Ibadan - Lagos – Abeokuta - Osogbo - Ilorin).
- The routes are then processed to produce the best possible tours with the shortest time.

- The distance and time of each iteration is then generated.
- Parameters are defined for both ACO & ABCO algorithms:

ACO:

- Number of cities = 5
- Pheromone evaporation rate = 0.5
- Number of iterations = 50

ABCO:

- Colony Size: 5
- Number of iterations = 50

2. Hard Constraints

- The hard constraint function is defined to check if a solution is truly the best tour that satisfies the traveling salesman problem.
- Hard constraint includes the completeness of the tour, the subtours, and the feasibility of the path.

3. Soft Constraints

- The soft constraint function is defined to check the soft constraint of the traveling salesman problem.
- Soft constraint includes minimization of tour length, balanced exploration and exploitation, computational efficiency.

4. Ant Colony Optimization and Artificial Bee Colony Optimization Algorithm

- The main loops runs for the specified number of iterations.

- The algorithm is evaluated for its fitness value, which include both hard and soft constraints.
- The best fitness value is updated if the new solution has a better fitness value
- The best tour is printed.
- The loop stops when the iterations reach its maximum indicating a feasible solution satisfying all constraints.
- After the algorithm finishes the final best iterations are printed, showing the best tour.

5. Performance Comparison

After running both algorithms, the performance of Ant Colony Optimization and Artificial Colony Optimization is compared based on their respective tours. The system takes into consideration each the constraints and outputs a detailed analysis of each algorithms results. This comparison serves as the basis of evaluating the effectiveness of each method.

6. Final Outputs

The best tour found by each algorithm is saved to the database. The system outputs both the final possible tours and summary of how well each algorithm performed the in terms of minimization of tour length.

4.3 SYSTEM REQUIREMENT

We look at the system requirements from the hardware and software application used for effective implementation of the new design.

4.3.1 Hardware requirement

For effective use of the new system, the minimum requirements for the hardware components are:

Table 1: Component Specification

COMPONENT	**SPECIFICATION**
Processor speeds	Intel(R) Pentium(R) CPU 4417U @ 2.30GHz
Ram Size	4.0 GB
Hard Disk	118 GB

4.3.2 Software requirement

For effective use of the new system, the minimum requirements for the software components are:

Table 2: Software requirements

SOFTWARE	**SPECIFICATION**
Operating System	Windows 10 Pro, 22H2
Code Editor IDE	Visual Studio Code
Programming Language	Python
Database	MySQL

4.3.3 Choice of Programming Language

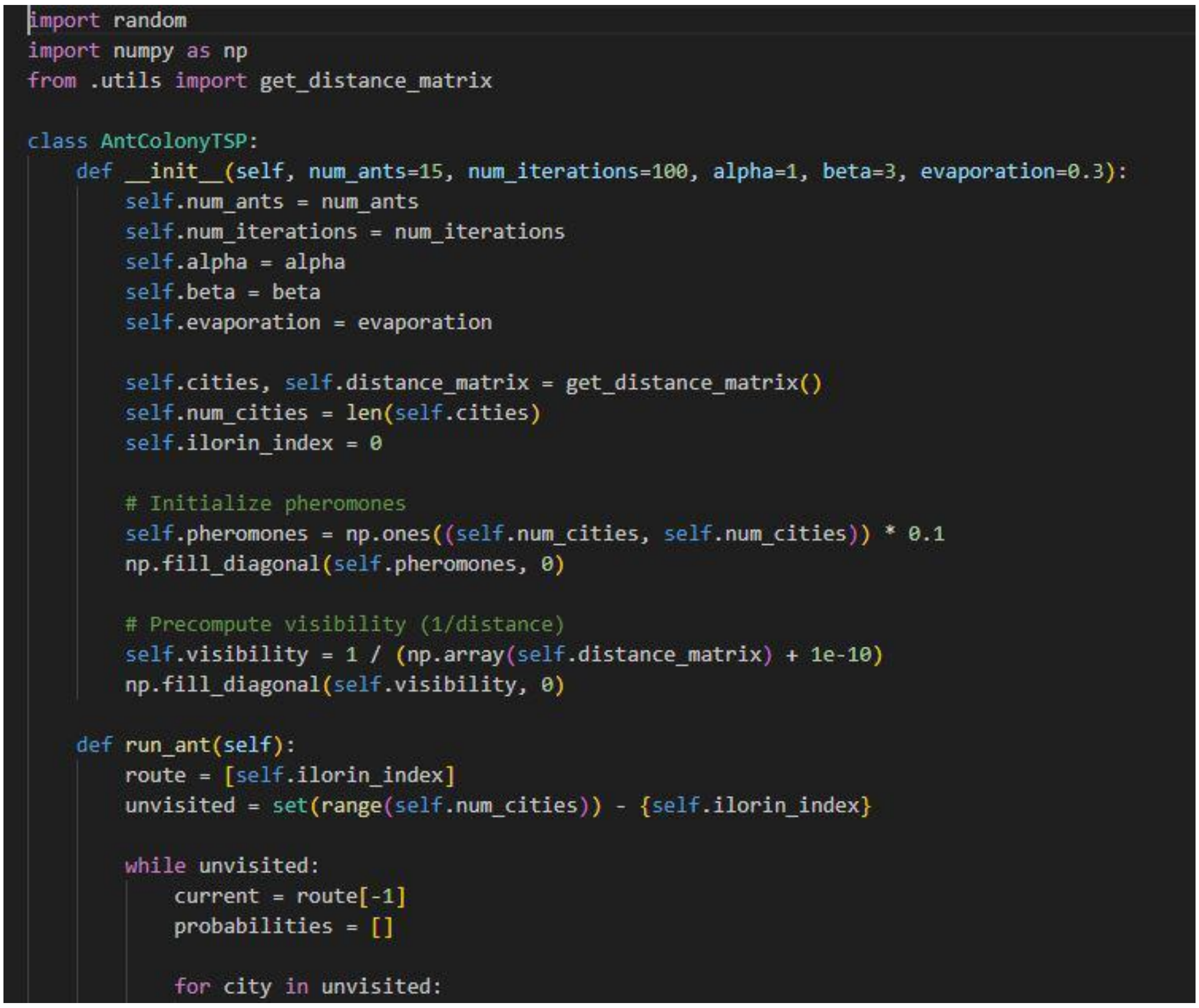
This system is developed in Visual Studio Code using Python programming language, a high-level language. With Visual Studio Code, you can install some plugins, analyze data and develop algorithms. Python is a high-level programming language that offers several advantages, including:

1. Easy to Learn: Python has a simple syntax and is relatively easy to learn, making it a great language for beginners.
2. High-Level Language: Python abstracts away many low-level details, allowing developers to focus on the logic of their program without worrying about memory management, etc.
3. Versatile: Python can be used for a wide range of applications, including web development, data analysis, machine learning, automation, and more.
4. Large Standard Library: Python has a vast collection of libraries and modules that make it easy to perform various tasks, such as file I/O, networking, and data analysis.
5. Dynamic Typing: Python is dynamically typed, which means that you don't need to declare the type of a variable before using it.
6. Cross-Platform: Python can run on multiple operating systems, including Windows, macOS, and Linux.

4.4 PROGRAM DOCUMENTATION

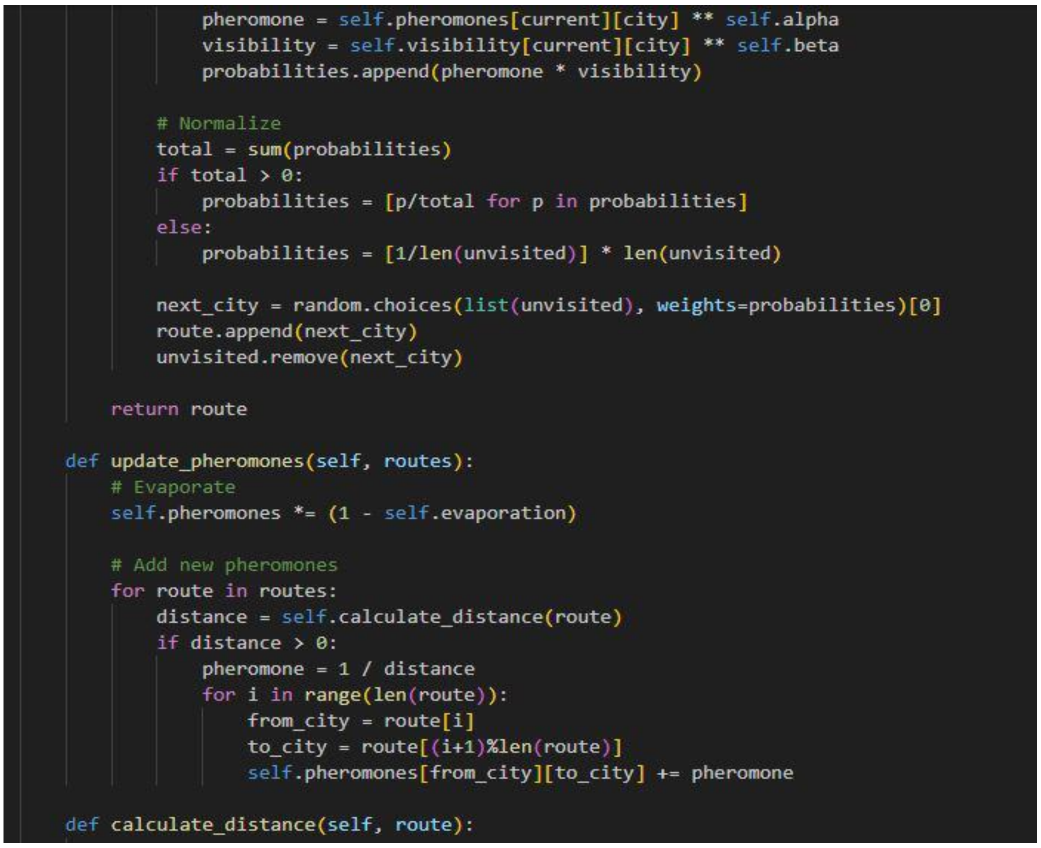
The python program for the traveling salesman problem to generate near optimal solution. The algorithm takes into consideration both hard constraints, such as the completeness of tour and soft constraints such as minimization of tour length. Here is the python Code for the algorithm below.

Fig 4.4 (1): ACO Code; where initialization takes place.



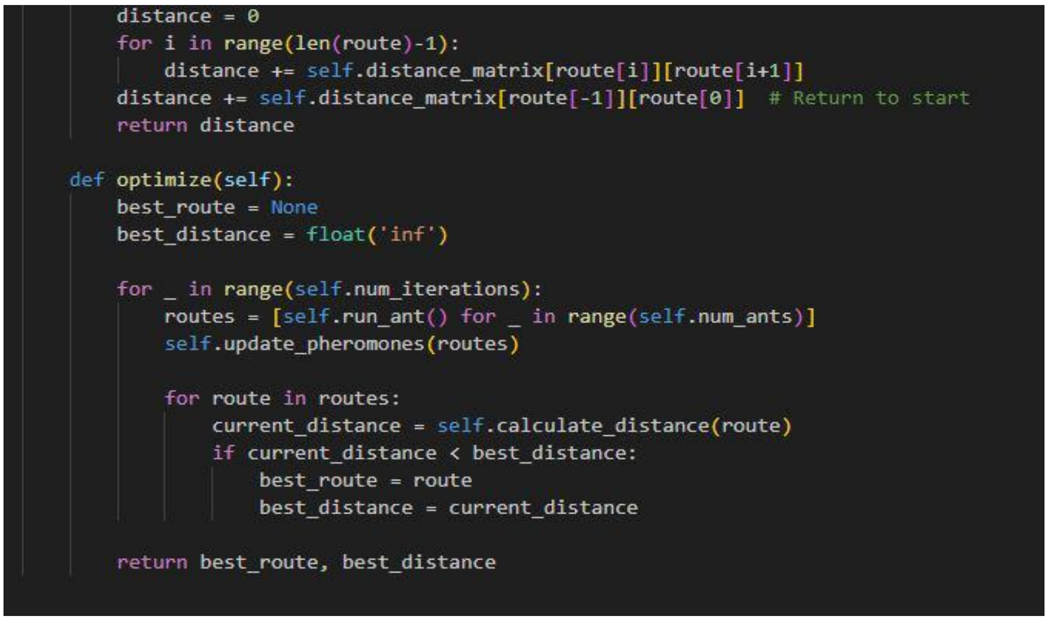
```python
import random
import numpy as np
from .utils import get_distance_matrix

class AntColonyTSP:
    def __init__(self, num_ants=15, num_iterations=100, alpha=1, beta=3, evaporation=0.3):
        self.num_ants = num_ants
        self.num_iterations = num_iterations
        self.alpha = alpha
        self.beta = beta
        self.evaporation = evaporation

        self.cities, self.distance_matrix = get_distance_matrix()
        self.num_cities = len(self.cities)
        self.ilorin_index = 0

        # Initialize pheromones
        self.pheromones = np.ones((self.num_cities, self.num_cities)) * 0.1
        np.fill_diagonal(self.pheromones, 0)

        # Precompute visibility (1/distance)
        self.visibility = 1 / (np.array(self.distance_matrix) + 1e-10)
        np.fill_diagonal(self.visibility, 0)

    def run_ant(self):
        route = [self.ilorin_index]
        unvisited = set(range(self.num_cities)) - {self.ilorin_index}

        while unvisited:
            current = route[-1]
            probabilities = []

            for city in unvisited:
```

Fig 4.4 (2): ACO Code; where initialization pheromones update takes place.



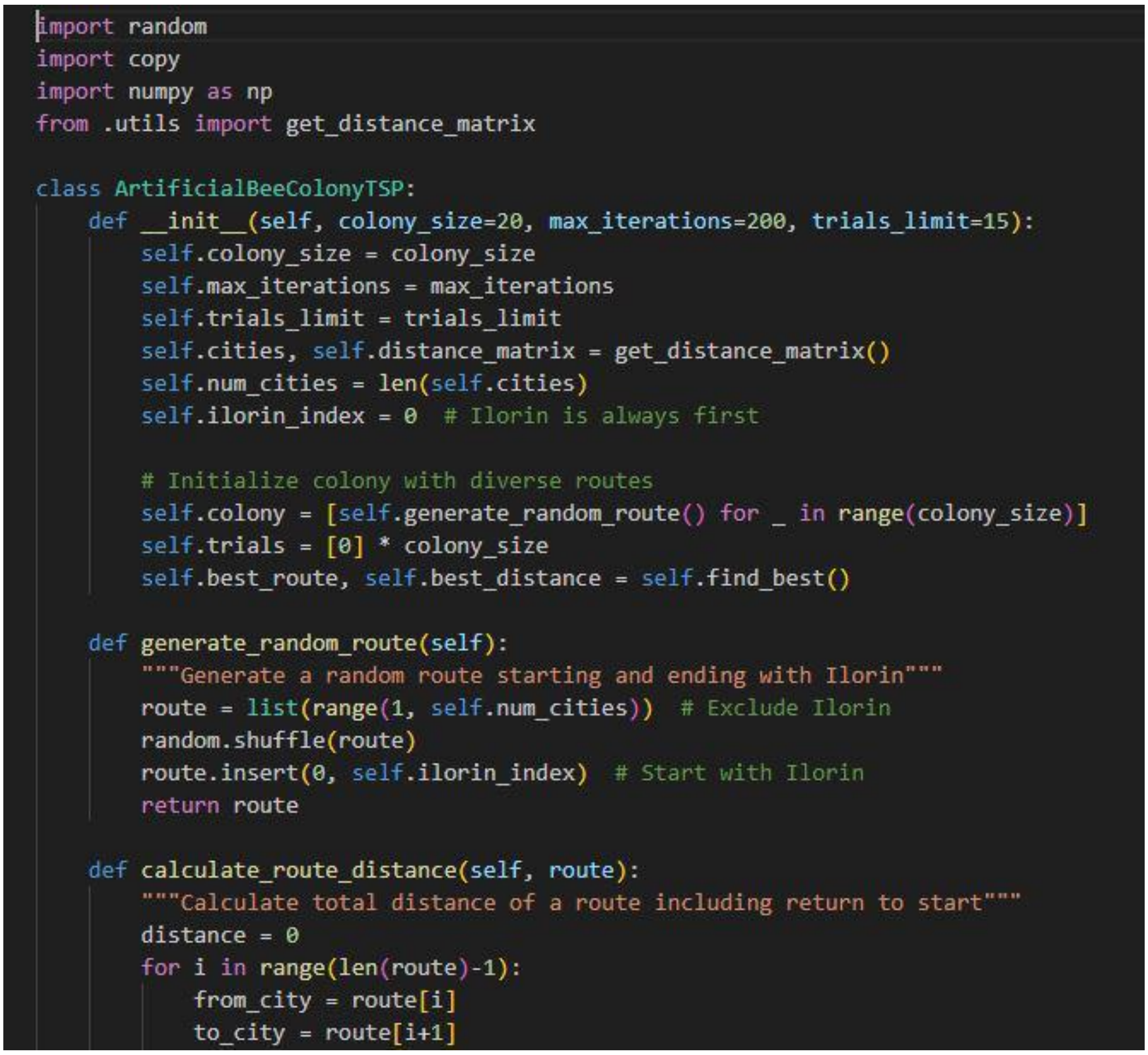
```
                pheromone = self.pheromones[current][city] ** self.alpha
                visibility = self.visibility[current][city] ** self.beta
                probabilities.append(pheromone * visibility)

            # Normalize
            total = sum(probabilities)
            if total > 0:
                probabilities = [p/total for p in probabilities]
            else:
                probabilities = [1/len(unvisited)] * len(unvisited)

            next_city = random.choices(list(unvisited), weights=probabilities)[0]
            route.append(next_city)
            unvisited.remove(next_city)

        return route

    def update_pheromones(self, routes):
        # Evaporate
        self.pheromones *= (1 - self.evaporation)

        # Add new pheromones
        for route in routes:
            distance = self.calculate_distance(route)
            if distance > 0:
                pheromone = 1 / distance
                for i in range(len(route)):
                    from_city = route[i]
                    to_city = route[(i+1)%len(route)]
                    self.pheromones[from_city][to_city] += pheromone

    def calculate_distance(self, route):
```

Fig 4.4 (3): ACO Code; where the final optimization takes place to update the best optimum solution.



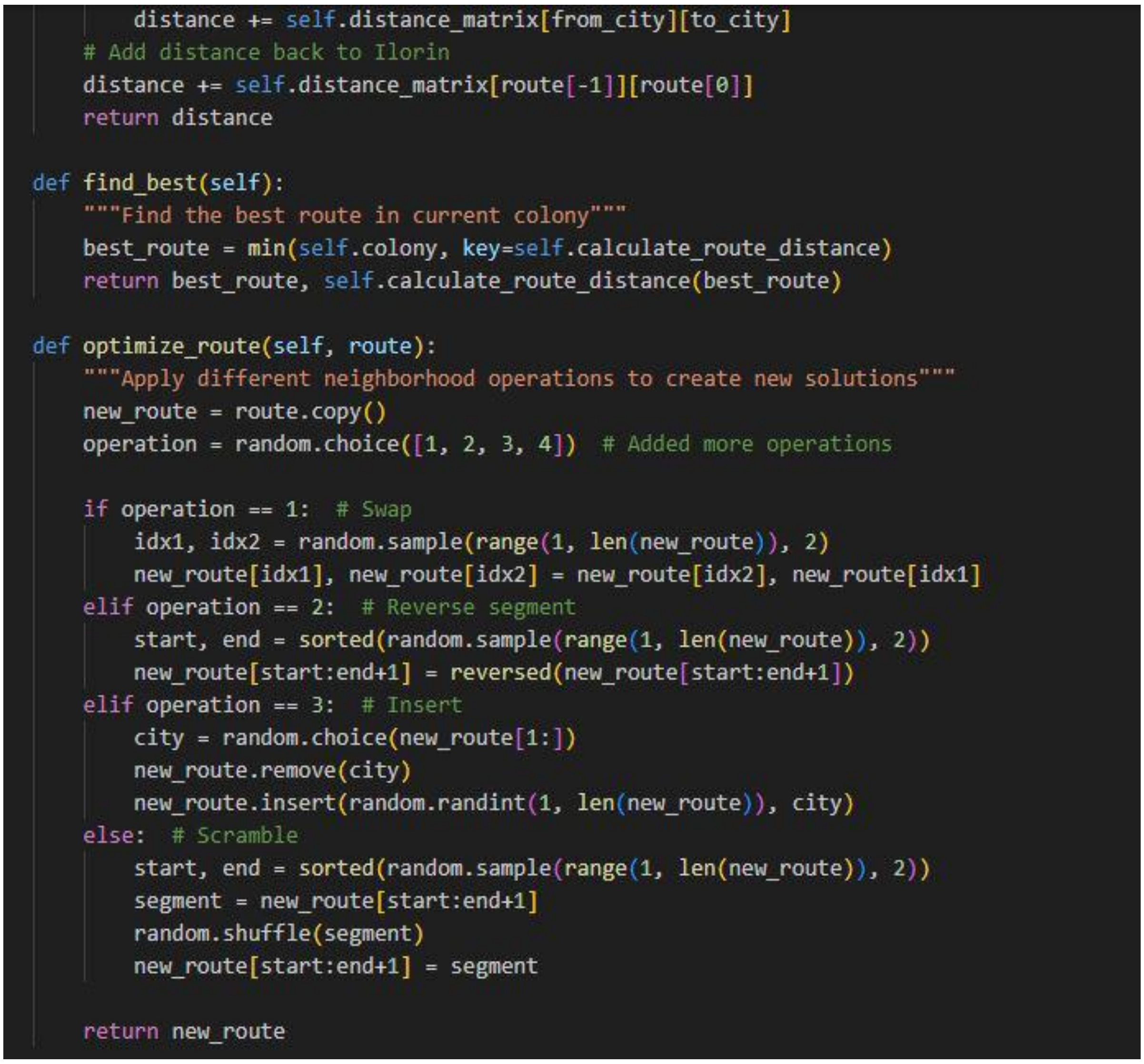
```
        distance = 0
        for i in range(len(route)-1):
            distance += self.distance_matrix[route[i]][route[i+1]]
        distance += self.distance_matrix[route[-1]][route[0]]  # Return to start
        return distance

    def optimize(self):
        best_route = None
        best_distance = float('inf')

        for _ in range(self.num_iterations):
            routes = [self.run_ant() for _ in range(self.num_ants)]
            self.update_pheromones(routes)

            for route in routes:
                current_distance = self.calculate_distance(route)
                if current_distance < best_distance:
                    best_route = route
                    best_distance = current_distance

        return best_route, best_distance
```

Fig 4.4 (4): ABCO Code; where initialization takes place.



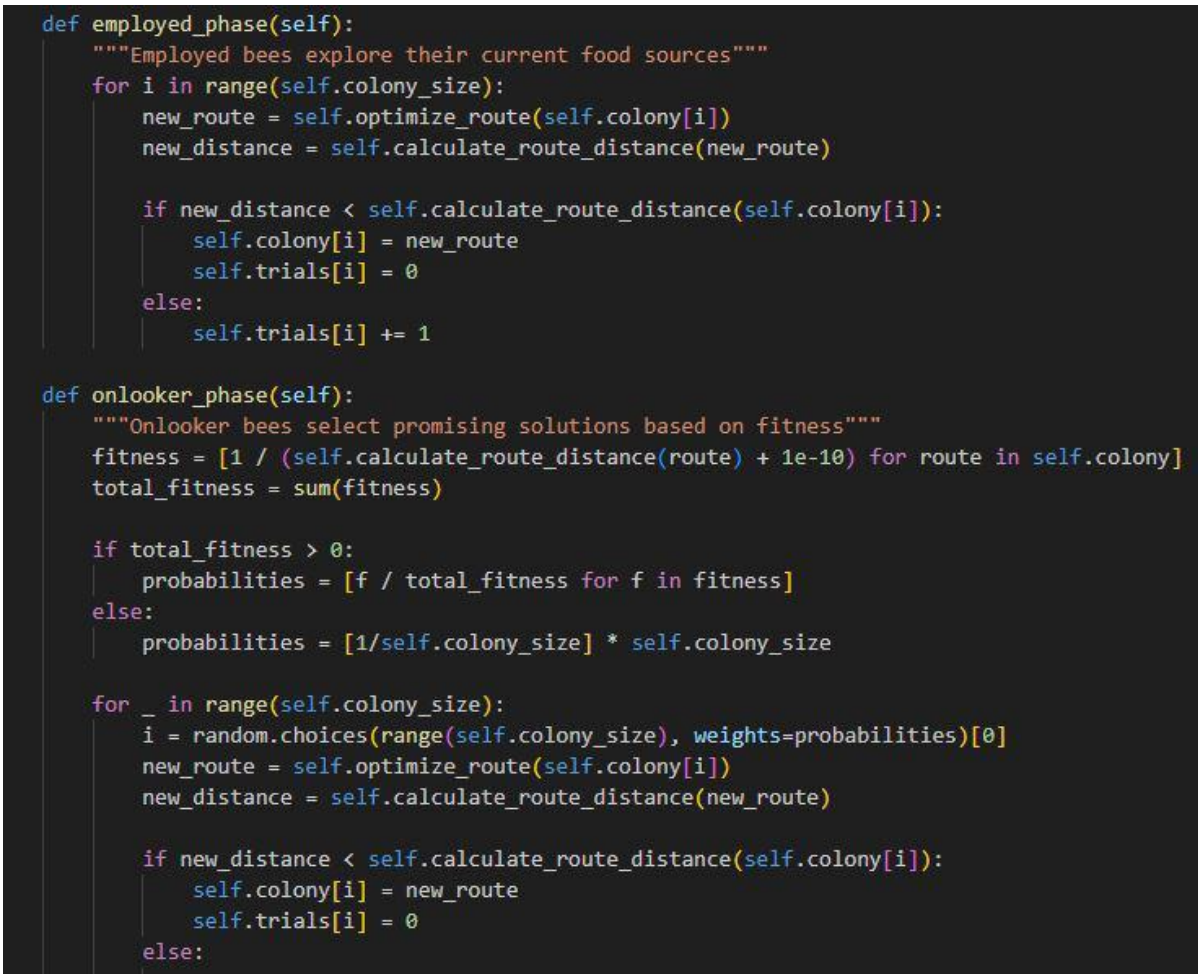
```python
import random
import copy
import numpy as np
from .utils import get_distance_matrix

class ArtificialBeeColonyTSP:
    def __init__(self, colony_size=20, max_iterations=200, trials_limit=15):
        self.colony_size = colony_size
        self.max_iterations = max_iterations
        self.trials_limit = trials_limit
        self.cities, self.distance_matrix = get_distance_matrix()
        self.num_cities = len(self.cities)
        self.ilorin_index = 0  # Ilorin is always first

        # Initialize colony with diverse routes
        self.colony = [self.generate_random_route() for _ in range(colony_size)]
        self.trials = [0] * colony_size
        self.best_route, self.best_distance = self.find_best()

    def generate_random_route(self):
        """Generate a random route starting and ending with Ilorin"""
        route = list(range(1, self.num_cities))  # Exclude Ilorin
        random.shuffle(route)
        route.insert(0, self.ilorin_index)  # Start with Ilorin
        return route

    def calculate_route_distance(self, route):
        """Calculate total distance of a route including return to start"""
        distance = 0
        for i in range(len(route)-1):
            from_city = route[i]
            to_city = route[i+1]
```

Fig 4.4 (5): ABCO Code; where optimization takes place.



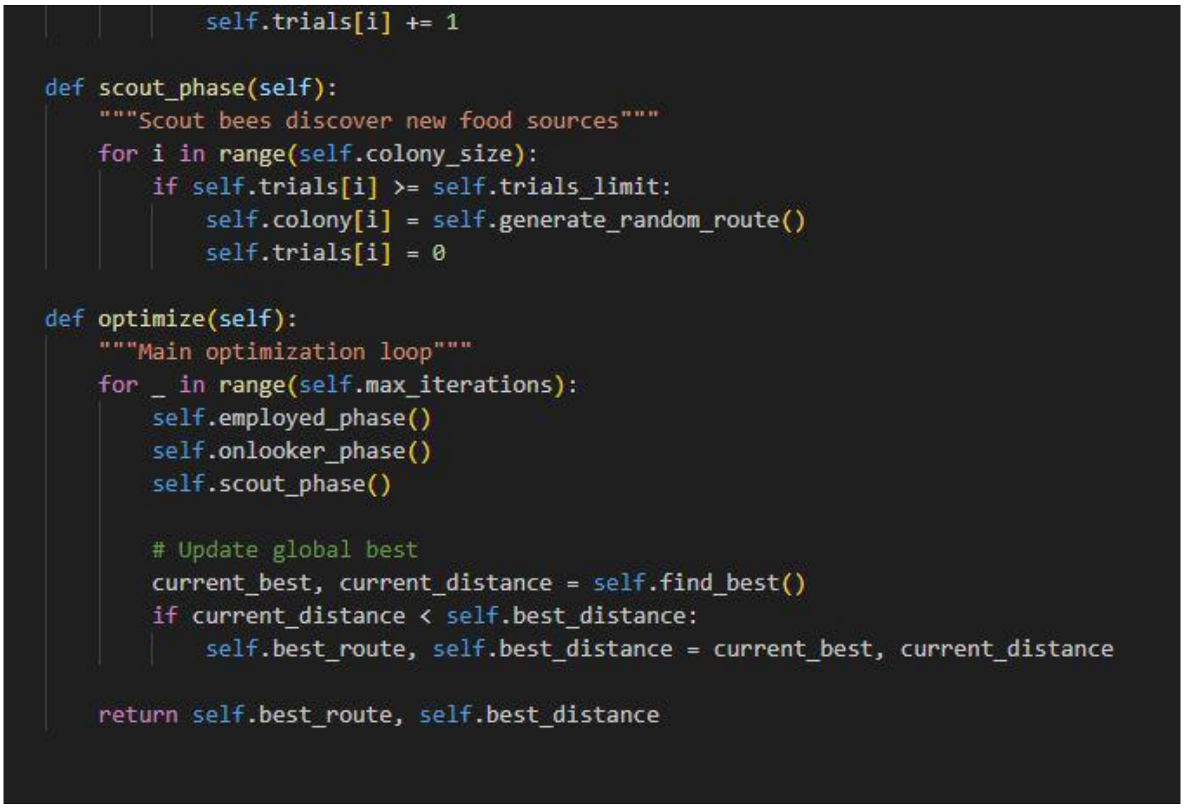
```
            distance += self.distance_matrix[from_city][to_city]
        # Add distance back to Ilorin
        distance += self.distance_matrix[route[-1]][route[0]]
        return distance

    def find_best(self):
        """Find the best route in current colony"""
        best_route = min(self.colony, key=self.calculate_route_distance)
        return best_route, self.calculate_route_distance(best_route)

    def optimize_route(self, route):
        """Apply different neighborhood operations to create new solutions"""
        new_route = route.copy()
        operation = random.choice([1, 2, 3, 4])  # Added more operations

        if operation == 1:  # Swap
            idx1, idx2 = random.sample(range(1, len(new_route)), 2)
            new_route[idx1], new_route[idx2] = new_route[idx2], new_route[idx1]
        elif operation == 2:  # Reverse segment
            start, end = sorted(random.sample(range(1, len(new_route)), 2))
            new_route[start:end+1] = reversed(new_route[start:end+1])
        elif operation == 3:  # Insert
            city = random.choice(new_route[1:])
            new_route.remove(city)
            new_route.insert(random.randint(1, len(new_route)), city)
        else:  # Scramble
            start, end = sorted(random.sample(range(1, len(new_route)), 2))
            segment = new_route[start:end+1]
            random.shuffle(segment)
            new_route[start:end+1] = segment

        return new_route
```

Fig 4.4 (6): ABCO Code; where the bee phase begin.



```
def employed_phase(self):
    """Employed bees explore their current food sources"""
    for i in range(self.colony_size):
        new_route = self.optimize_route(self.colony[i])
        new_distance = self.calculate_route_distance(new_route)

        if new_distance < self.calculate_route_distance(self.colony[i]):
            self.colony[i] = new_route
            self.trials[i] = 0
        else:
            self.trials[i] += 1

def onlooker_phase(self):
    """Onlooker bees select promising solutions based on fitness"""
    fitness = [1 / (self.calculate_route_distance(route) + 1e-10) for route in self.colony]
    total_fitness = sum(fitness)

    if total_fitness > 0:
        probabilities = [f / total_fitness for f in fitness]
    else:
        probabilities = [1/self.colony_size] * self.colony_size

    for _ in range(self.colony_size):
        i = random.choices(range(self.colony_size), weights=probabilities)[0]
        new_route = self.optimize_route(self.colony[i])
        new_distance = self.calculate_route_distance(new_route)

        if new_distance < self.calculate_route_distance(self.colony[i]):
            self.colony[i] = new_route
            self.trials[i] = 0
        else:
```

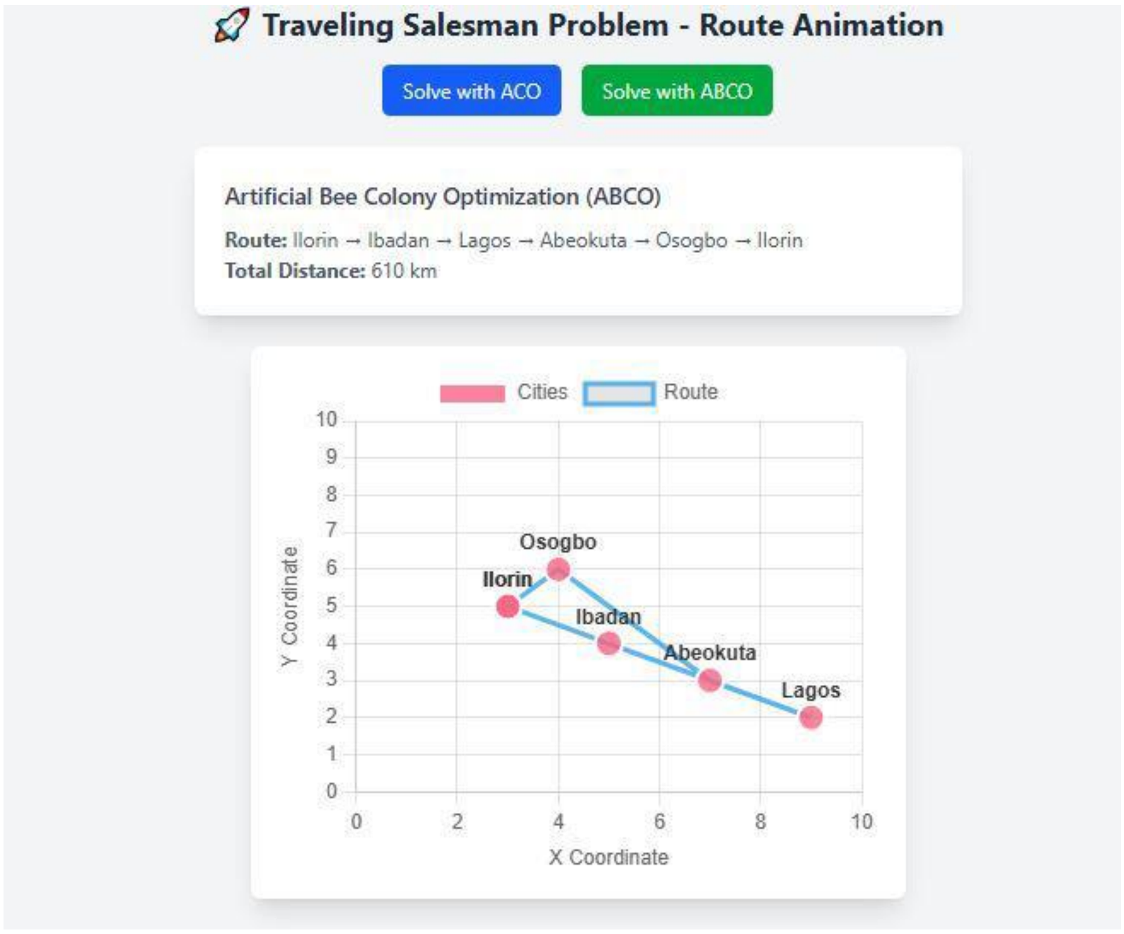
Fig 4.4 (7): ABCO Code; where the bee phases ends and final optimization takes place to update the best optimum solution.



```
                self.trials[i] += 1

    def scout_phase(self):
        """Scout bees discover new food sources"""
        for i in range(self.colony_size):
            if self.trials[i] >= self.trials_limit:
                self.colony[i] = self.generate_random_route()
                self.trials[i] = 0

    def optimize(self):
        """Main optimization loop"""
        for _ in range(self.max_iterations):
            self.employed_phase()
            self.onlooker_phase()
            self.scout_phase()

            # Update global best
            current_best, current_distance = self.find_best()
            if current_distance < self.best_distance:
                self.best_route, self.best_distance = current_best, current_distance

        return self.best_route, self.best_distance
```

4.5 INTERFACE DESIGN

This section presents the user interface design developed for the system. Each interface is designed with user friendliness and usability ensuring a smooth user experience. The images below shows the key screen of the system, including navigation elements, display outputs and highlighting how users interact with the application.

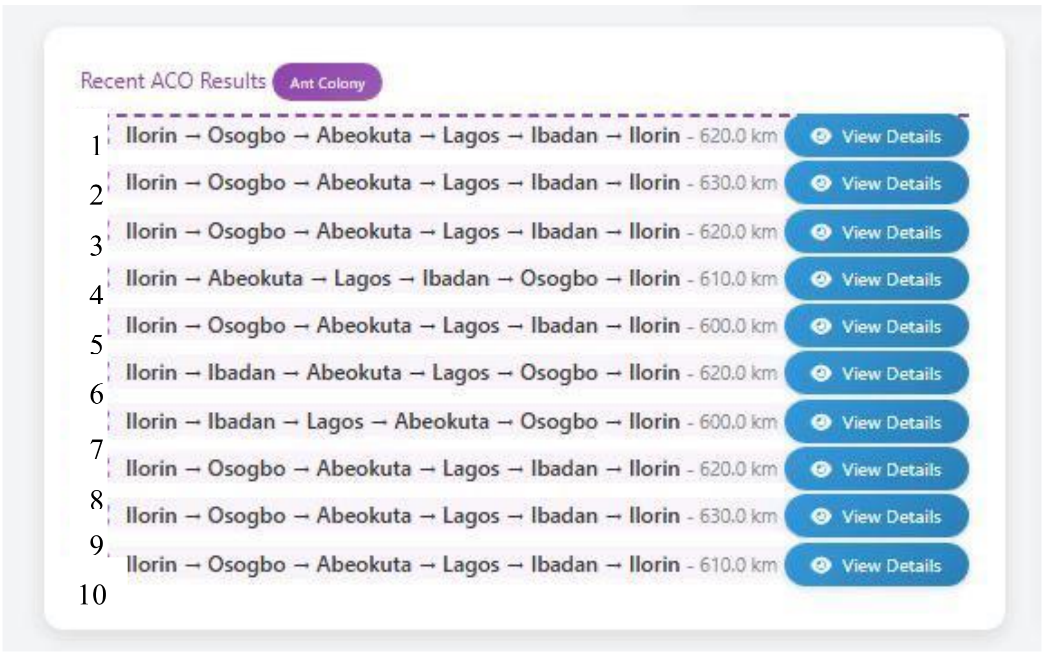


Artificial Bee Colony Optimization (ABCO)

Route: Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin

Total Distance: 610 km

Fig 4.5.1: Route Animation



Recent ACO Results Ant Colony

1 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 620.0 km View Details

2 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 630.0 km View Details

3 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 620.0 km View Details

4 Ilorin → Abeokuta → Lagos → Ibadan → Osogbo → Ilorin - 610.0 km View Details

5 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 600.0 km View Details

6 Ilorin → Ibadan → Abeokuta → Lagos → Osogbo → Ilorin - 620.0 km View Details

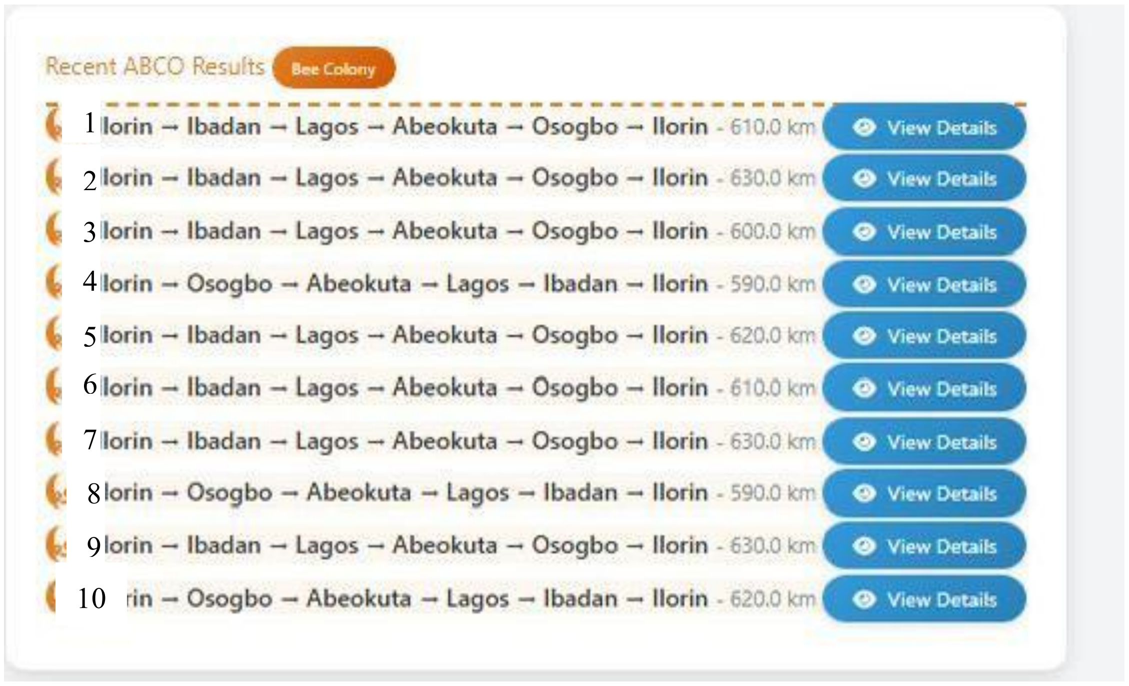
7 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 600.0 km View Details

8 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 620.0 km View Details

9 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 630.0 km View Details

10 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 610.0 km View Details

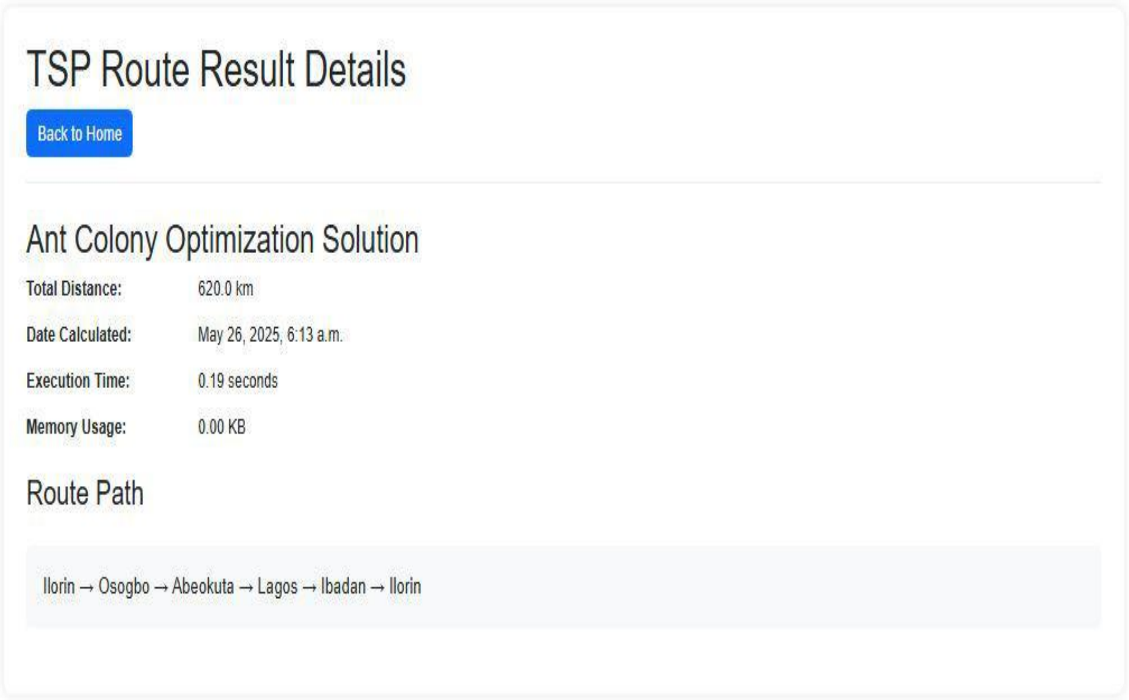
Fig 4.5.2: ACO Route iterations


Recent ABCO Results Bee Colony

1 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 610.0 km View Details
2 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 630.0 km View Details
3 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 600.0 km View Details
4 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 590.0 km View Details
5 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 620.0 km View Details
6 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 610.0 km View Details
7 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 630.0 km View Details
8 Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 590.0 km View Details
9 Ilorin → Ibadan → Lagos → Abeokuta → Osogbo → Ilorin - 630.0 km View Details
10 rin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin - 620.0 km View Details

Fig 4.5.3: ABCO Route iterations

Fig 4.5.4: Result of a TSP Route Analysis based on distance, time and memory usage.



TSP Route Result Details

Back to Home

Ant Colony Optimization Solution

Total Distance:	620.0 km
Date Calculated:	May 26, 2025, 6:13 a.m.
Execution Time:	0.19 seconds
Memory Usage:	0.00 KB

Route Path

Ilorin → Osogbo → Abeokuta → Lagos → Ibadan → Ilorin

4.6 RESULTS AND DISCUSSION

It's experimented with the traveling salesmen problem data to determine how well the Ant Colony Optimization and Artificial Bee Colony Optimization solved the traveling salesmen problem. It is solved by utilizing the ACO & ABCO by taking into account both hard and Soft constraints, generate new solution, evaluate their fitness value and update the best solution. The result makes it easier to keep track of the **algorithm's progress which is as follows:**

TABLE 3: RESULT SUMMARY FOR ANT COLONY OPTIMIZATION ALGORITHM

Iteration	Route path	Distance Taken (Tour length)	Time Taken (Sec)	Memory Usage
1.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	620.0km	0.19	0.00kb
2.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	630.0km	0.19	0.00kb
3.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	620.0km	0.20	0.00kb
4.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	620.0km	0.21	0.00kb
5.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	600.0km	0.19	0.00kb
6.	Ilorin(173km)_Ibadan(123km)_Abeokuta(19km)_Lagos(224km)_Osogbo(115km)_Ilorin	620.0km	0.22	12.00kb
7.	**Ilorin(173km)_Ibadan(123km)_Abeokuta(19km)_Lagos(224km)_Osogbo(115km)_Ilorin**	**600.0km**	**0.18**	**0.00kb**
8.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	610.0km	0.16	4.00kb
9.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	630.0km	0.17	16.00kb
10.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	610.0km	0.17	152.00kb

Ant Colony Optimization results discussion: The first ACO output starts with iteration 1 and continues to iteration 50, each iteration were run and the best 10 iteration was recorded which includes the route path distance, distance, time taken and memory usage. The best output tour as indicated in the table above is iteration 7 with the distance of 600.0 km, the execution time of 0.18 secs and memory usage of 0.00kb.

TABLE 4: RESULT SUMMARY FOR ARTIFICIAL BEE COLONY OPTIMIZATION ALGORITHM

Iteration	Route path	Distance Taken(tour length)	Time Taken(Sec)	Memory Usage
1.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	610.0km	0.25	0.00kb
2.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	630.0km	0.28	4.00kb
3.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	600.0km	0.27	0.00kb
4.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	590.0km	0.27	0.00kb
5.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	620.0km	0.26	0.00kb
6.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	610.0km	0.35	0.00km
7.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	630.0km	0.25	0.00kb
8.	**Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin**	**590.0km**	**0.25**	**0.00kb**
9.	Ilorin(173km)_Ibadan(131km)_Lagos(19km)_Abeokuta(211km)_Osogbo(115km)_Ilorin	630.0km	0.21	12.00kb
10.	Ilorin(115km)_Osogbo(211km)_Abeokuta(19km)_Lagos(131km)_Ibadan(173km)_Ilorin	620.0km	0.22	28.00kb

Artificial Bee Colony Optimization results discussion: The first ABCO output starts with iteration 1 and continues to iteration 50, each iteration was run and the best 10 iteration was recorded which includes the route path, distance, time taken and memory usage. The best output tour as indicated in the table above is iteration 8 with the distance of 590.0 km, the execution time of 0.25 secs and memory usage of 0.00kb.

Analysis of Ant Colony Optimization result for 10 iteration of the best tours of the TSP, in the first iteration, the algorithm returns a result with a distance of 620.0km, total execution time of 0.19 secs and 0.00kb memory usage. The algorithm proceeds through the iterations to search for near optimal solution, aim to satisfy the soft constraint (minimization of tour length). The second iteration returns a result with a distance of 630.0km, total execution time of 0.19 secs and 0.00kb memory usage. The third iteration returns a result with a distance of 620.0km, total execution time of 0.20 secs and 0.00kb memory usage. The fourth iteration returns with a result with a distance of 620.0km, total execution time of 0.21 secs and 0.00kb memory usage. The fifth iteration returns with a result with a distance of 600.0km, total execution time of 0.19 secs and 0.00kb memory usage. The sixth iteration returns with a result with a distance of 620.0km, total execution time of 0.22 secs and 12.00kb memory usage. The seventh iteration returns with a result with a distance of 600.0km, total execution time of 0.18 secs and 0.00kb memory usage. The eighth iteration returns with a result with a distance of 620.0km, total execution time of 0.16 secs and 4.00kb memory usage. The ninth iteration returns with a result with a distance of 630.0km, total execution time of 0.17 secs and 16.00kb memory usage while the tenth iteration returns with a distance of 610.0km, total execution time of 0.17 secs and 152.00kb memory usage. In these 10 iterations, it shows that the 7th iteration is the best four with a distance of 600.0km, total execution time of 0.18 secs and 0.00kb memory usage.

Analysis of Artificial Bee Colony Optimization result for 10 iteration of the best tours of

the TSP, in the first iteration, the algorithm returns a result with a distance of 610.0km, total execution time of 0.25 secs and 0.00kb memory usage. The algorithm proceeds through the iterations to search for near optimal solution, aim to satisfy the soft constraint (minimization of tour length). The second iteration returns a result with a distance of 630.0km, total execution time of 0.28 secs and 4.00kb memory usage. The third iteration returns a result with a distance of 600.0km, total execution time of 0.27 secs and 0.00kb memory usage. The fourth iteration returns with a result with a distance of 590.0km, total execution time of 0.27 secs and 0.00kb memory usage. The fifth iteration returns with a result with a distance of 620.0km, total execution time of 0.26 secs and 0.00kb memory usage. The sixth iteration returns with a result with a distance of 610.0km, total execution time of 0.35 secs and 0.00kb memory usage. The seventh iteration returns with a result with a distance of 630.0km, total execution time of 0.25 secs and 0.00kb memory usage. The eighth iteration returns with a result with a distance of 590.0km, total execution time of 0.25 secs and 0.00kb memory usage. The ninth iteration returns with a result with a distance of 630.0km, total execution time of 0.21 secs and 12.00kb memory usage while the tenth iteration returns with a distance of 620.0km, total execution time of 0.22 secs and 28.00kb memory usage. In these 10 iterations, it shows that the 8th iteration is the best four with a distance of 590.0km, total execution time of 0.25 secs and 0.00kb memory usage.

TABLE 5: BEST ITERATION SUMMARY

Algorithm	**Best iteration**	**Distance (Tour length)**	**Time Taken (Sec)**	**Memory Usage(Kb)**
Ant Colony Optimization	7th	600.0km	0.18	0.00
Artificial Bee Colony Optimization	8th	590.0km	0.25	0.00

PERFORMANCE COMPARISON

Soft Constraint Satisfaction (minimization of tour length): Artificial Bee Colony Optimization wins because it achieves a shorter distance of (590.0km) compared to Ant Colony optimization of (600.0km) which means it satisfies the soft constraints better.

Execution Time: ACO is slightly faster in its best iteration (0.18 secs) than ABCO (0.25 secs) although the difference is minimal (0.07 secs).

Memory Usage: There is no difference between the two algorithms as they both produced the same output of memory usage (0.00kb).

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 SUMMARY

This project focused on solving the Traveling Salesman Problem (TSP) using two bio-inspired metaheuristic algorithms, Ant Colony Optimization (ACO) and Artificial Bee Colony Optimization (ABCO). TSP, a classical NP-hard problem, requires determining the shortest route that visits a set of cities exactly once and returns to the starting point. Due to its computational complexity, exact methods are often impractical for large instances, hence the need for heuristic-based approaches.

The project involved the adaptation and implementation of both ACO and ABCO algorithms to solve a TSP instance involving five major Nigerian cities (Ilorin, Abeokuta, Lagos, Ibadan, and Osogbo). The algorithms were developed using Python and integrated into a Django framework, with MySQL used for data storage. Both algorithms were evaluated based on their ability to satisfy hard constraints (e.g., visiting all cities, no sub tours) and soft constraints (e.g., minimizing distance, execution time, and computational efficiency).

Performance comparison showed that ABCO achieved a shorter tour length of 590 km, demonstrating superior solution quality, while ACO outperformed in convergence speed with a faster execution time of 0.18 seconds. Memory usage was minimal for both algorithms.

In conclusion, the study demonstrated the practicality of swarm intelligence algorithms in solving complex optimization problems like TSP. It provided insights into their comparative strengths, establishing a foundation for further research and application in logistics, route planning, and other domains requiring efficient pathfinding solutions.

5.2 CONCLUSION

The comparative evaluation of the Ant Colony Optimization and Artificial Bee Colony Optimization algorithm provides valuable insight into solving the Traveling Salesman problem. By considering the solution quality, convergence speed e.t.c. Users can identify the most suitable algorithm to solve the TSP with the appropriate data and objective.

The overall better algorithm is the Ant Colony optimization algorithm, its convergence speed (0.18 secs) means it satisfies the soft constraints better than the Artificial Bee Colony Optimization algorithm (0.25 secs) even though its solution quality is slightly higher than that of the ABCO. The algorithm's performance was evaluated based on the result obtained from the 10 iterations of each algorithm and the resources utilized.

5.3 RECOMMENDATIONS

Based on the findings from this research, the following recommendations are made:

1. Adopt ABCO for real-world TSP applications such as route planning in logistics, especially in small to medium-scale transportation networks like the case study of Nigerian cities.
2. Further research should explore hybrid models, combining the strengths of ACO and ABCO to address the weaknesses of each and potentially improve performance for more complex or large-scale TSP instances.
3. Optimization models should be tailored to specific problem sizes. While ABCO works well for small datasets, ACO may still be suitable for very large datasets with proper parameter tuning.

4. Further research should explore hybrid models, combining the strengths of ACO and ABCO to address the weaknesses of each and potentially improve performance for more complex or large-scale TSP instances.
5. Practical applications in logistics, delivery services, and supply chain management should consider ABCO-based routing models to improve operational efficiency and cost-effectiveness.

REFERENCES

Akay, B., Aydogan, E., & Karacan, L. (2012). *2-opt based artificial bee colony algorithm for solving traveling salesman problem. AWER Procedia — Information Technology & Computer Science, 1, 666-672*. https://doi.org/10.1016/j.procs.2012.05.104

Dahan, F., El Hindi, K., Mathkour, H., & AlSalman, H. (2019). *Dynamic flying ant colony optimization (DFACO) for solving the traveling salesman problem. Sensors, 19*(8), 1837. https://doi.org/10.3390/s19081837

Dhanasekar, S., Dash, S. K., & Uthaman, N. (2022). *A branch and bound algorithm to solve travelling salesman problem (TSP) with uncertain parameters. Mathematics and Statistics, 10(2), 358–365*. https://doi.org/10.13189/ms.2022.100210

Ella, J., & McMullen, P. R. (2004). *Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics, 18(1), 41–48.* https://doi.org/10.1016/j.aei.2004.07.001

Jamaluddin, S. H., Mohd Naziri, N. A. H., Mahmud, N., & Muhammat Pazil, N. S. (2022). *Solving the Traveling Salesman Problem by Using Artificial Bee Colony Algorithm. Journal of Computing Research and Innovation, 7(2), 121-131.* https://doi.org/10.24191/jcrinn.v7i2.295

Karaboga, D., & Gorkemli, B. (2019). *Solving Traveling Salesman Problem by Using Combinatorial Artificial Bee Colony Algorithms. International Journal on Artificial Intelligence Tools, 28(1), 1950004.* https://doi.org/10.1142/S0218213019500040

Kiran, M. S., & Beskirli, M. (2024). *A New Approach Based on Collective Intelligence to Solve Traveling Salesman Problems. Biomimetics, 9(2), 118.*

Liu, H., Lee, A., Lee, W., & Guo, P. (2023). *DAACO: Adaptive dynamic quantity of ant ACO algorithm to solve the traveling salesman problem. Complex & Intelligent Systems*. https://doi.org/10.1007/s40747-022-00949-6

Rufai, K. I., Usman, O. L., Olusanya, O. O., & Adedeji, O. B. (2021). *Solving travelling salesman problem using an improved ant colony optimization algorithm. University of Ibadan Journal of Science and Logics in ICT Research, 6*(2), 158–170.

Skinderowicz, R. (2022). *Improving ant colony optimization efficiency for solving large TSP instances. Applied Soft Computing, 114, 108653.* https://doi.org/10.1016/j.asoc.2022.108653

Xing, Z., & Tu, S. (2020). *A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem. IEEE Access, 8, 108418–108428.*

Yang, J., Shi, X., Liang, Y., & (2008). *An ant colony optimization method for the generalized traveling salesman problem. Progress in Natural Science, 18(7), 1417–1422.* https://doi.org/10.1016/j.pnsc.2008.03.028