Advanced Metaheuristics for Optimization

Thesis for the Degree of Doctor of Philosophy (PhD)

by: Anahita Sabagh Nejad

Supervisor: Dr. Gabor Fazekas

UNIVERSITY OF DEBRECEN
Doctoral Council of Natural Sciences and Information Technology
Doctoral School of Informatics
Debrecen, 2024

Hereby I declare that I prepared this thesis within the Doctoral Council of Natural Sciences and Information Technology, Doctoral School of Informatics, University of Debrecen in order to obtain a Ph.D. Degree in Informatics at Debrecen University.
The results Published in the thesis are not reported in any other PhD theses.
Debrecen, 2024

.....
Signature

Hereby I confirm that Anahita Sabagh Nejad candidate conducted her studies with my supervision within the Applied Information Technology and its Theoretical Background Doctoral Program of the Doctoral School of Informatics, University of Debrecen between 2016 and 2021. The independent studies and research work of the candidate significantly contributed to the results published in the thesis.
I also declare that the results published in the thesis are not reported in any other theses.
I suppose the acceptance of the thesis.

.....
Signature

Advanced Metaheuristics for Optimization

Dissertation submitted in partial fulfillment of the requirements for the doctoral (Ph.D.) degree
in informatics

Written by: Anahita Sabagh Nejad

certified computer scientist
Prepared in the framework of the Doctoral School of Informatics of the University of Debrecen (Applied Information Technology and its Theoretical Background doctoral programme)

Dissertation advisor: Dr. Gabor Fazekas

The official opponents of the dissertation:

Dr. .....
Dr. .....

The evaluation committee:

chairperson: Dr. .....
members: Dr. .....
Dr. .....
Dr. .....
Dr. .....

The date of the dissertation defense: 2024 .....

Contents

Chapter 1

Introduction

The topic of my dissertation is a Traveling salesman problem (TSP) which is solved by metaheuristics. Solving this optimization problem is important as it is an NP-hard and can't be solved in a polynomial time. Optimization problems can be discrete or continuous. We want to explain some basic theories, terminologies, and ideas. Then, I summarize some explanations about some of the bio-inspired algorithms. The pseudocodes and methods will be explained in the sections and sub-sections. I have written a conference paper and two articles in the field of bio-inspired algorithms. I mainly focus on the whale optimization algorithm in my articles. There are so many algorithms and mathematical test functions but I considered the statistical values as benchmarks, specifically average and minimum values (min). The concentration was on the Whale algorithm as a bio-inspired algorithm, K- means, as a clustering technique, and the Birch algorithm as another method for solving TSP that I explain in the text. In the proposed methods that I will explain in Chapter 5, I have modified the whale optimization algorithm to solve the traveling salesman problem.

1.1 Overview

In the domain survey, I discussed the graphs because TSP is a graph with a Hamiltonian path that we will explain in the second chapter. There are some algorithms like VRP or some algorithms that can be simulated as TSP to solve some real-world problems. Some data mining techniques are introduced like association rules, clustering (unsupervised learning), classification(supervised learning), regression, and grid-based methods with some of their algorithms as examples. Still, the main focus is on clustering, specifically partitioning-based and hierarchy-based methods. I solved TSP using a combinatorial method using the whale algorithm and k-means which is a partitioning-based clustering algorithm. In this thesis, many bio-inspired algorithms are also explained which are among the most famous algorithms in the field of metaheuristics and can be used to solve TSP and other optimization problems.

1.2 The Organization of the Thesis

This document can be partitioned into some sections. These sections provide the general required knowledge where some basic information about data mining techniques, and some of the swarm intelligence algorithms have been explained. After the introduction in Chapter 1, the graph theory and some other related algorithms are summarized in Chapter 2.
In Chapter 3, some data mining methods and techniques are summarized. Chapter 4 is dedicated to Metaheuristics and begins with the Genetic algorithm and follows with many bio-inspired algorithms. The whale algorithm is also explained in this chapter.
Chapter 5 presents a summarization of the related papers and the newfound methods. Chapter 6 is the main statement and conclusion of the dissertation.

Chapter 2

Graph Theory definitions

A graph is a non-empty set of finite Vertices (V) [1] along with a set of Edges (E) of d elements. The mathematical notion of a graph is G (V, E). The cardinality of a graph is the number of Vertices (nodes), and the degree of a vertice or deg(v) is the number of edges that are connected to that vertice [2]. Some graphs are isomorphic which means the two graphs are the same. Adjacency list in the graphs means the summarization of the graph in the form of some connected neighbor vertices. An adjacency matrix is a matrix that shows the connection between the nodes. If there is a connection, it assigns 1, otherwise 0. The tree structure helps in searching for the programming essence. In graph theory, multi-graphs have loops and parallel edges, so we can say that the regular graph is a type of multi-graph. A regular graph is a type of complete graph, and the complete graph is a Hamiltonian.

Euler path [3] is a path in a finite undirected graph that covers every edge just once and allows revisiting nodes. If every node of this path has an even degree, it is called the Euler Circuit.

The process of edge decomposition to Hamiltonian circuits is called Hamiltonian decomposition. Hamilton Cycle (traceable path) (HC) is a path in a (un)directed graph that visits every $v \in V$ just once. A graph G of the Hamilton Cycle is also called the Hamiltonian

graph. If all the Vertices are connected this way, the graph will be Hamiltonian-connected. A Hamiltonian path completes by adding one more Edge to create a cycle (circuit). Where these paths and cycles exist the problem becomes NP-complete. The Hamiltonian cycle(HC) is related to TSP and it has many applications in engineering (for example routing for networks), and mathematics (for example timing), and that's why resolving this problem is important in computer science. Some of the theorems [3] provide sufficient conditions for a graph to be Hamiltonian [such as Ore's (1961) and Dirac's (1952) theorems], but those conditions are complicated and not applicable. Some complicated theorems like Ore's (1961) and Dirac's (1952) introduce some conditions for an HC graph. The Euler Cycle also has sufficient and applicable conditions and algorithms to find the cycles. Here, we have some definitions:

Definition 1 For an Undirected G where $v \in V$ we can define a surplus (v)such that sur(v) = deg (v) – 2 [3]. It means the sur(v) is the number of incident edges to v in a graph G (V, E) [4] that must be deleted so that the deg(v)=2 [3]. For |E| is the number of edges and |V| is the number of vertices [3] we have:

$$\sum_{x \in V} sur(x) = \sum_{x \in V} (deg(x) - 2) = \sum_{x \in V} deg(x) - \sum_{x \in V} 2 = 2(|E| - |V|) \tag{2.0.1}$$

Dirac's theorem A simple graph G has a Hamiltonian cycle if the degree of a vertex is minimum $\left\lfloor \frac{n}{2} \right\rfloor$ when n is the number of vertices [3].

Ore's theorem A simple graph G has a Hamiltonian cycle if the sum of the degrees for any two non-adjacent vertices is greater or equal to n (where n is the number of vertices)[3].

To find an HC, the edges that form it must be chosen so that any extra edges are deleted based on definition 2. This way, all the non-HC edges become deleted from the graph[3].

This process is expressed in the Algorithm 1 below:

Definition 2: A deletion of a graph G is a random way of

deleting edges that are not incident to vertices with zero surpluses. The deletion algorithm keeps deleting until it can't remove more edges. The deleted edges are called surplus edges [3]. This process is expressed in the next figure: [3]

Algorithm 1

```
Input: a graph G(V, E)

Output: a graph G' without surplus edges

deletion(G)

{

        for each vertex  v in V

                for each edge e incident to v

                        if e is a surplus  edge

                                remove e

}
```

Figure 2.0.1: Algorithm 1

2.1 TSP Problem

The traveling salesman problem (TSP) is a famous combinatorial optimization problem that has model characters in different fields such as Mathematics, Computer Science, and Operations Research [5].
Heuristics, linear programming, branch and bound, Lagrangean relaxation, Lin- Kernighan, simulated annealing, and the field of polyhedral combinatorics were formulated for the TSP and used to solve practical problem instances in 1954 by Dantzig, Fulkerson, and Johnson. When the theory of NP-completeness developed, the TSP was one of the first problems to be proven NP-hard by Karp in 1972.
If a traveling salesman wishes to visit exactly once each of a list of m cities [6] (where the cost of traveling from city i to city j is (c_{ij}) and then return to the home city, finding a route with minimum cost becomes a challenge, as it is explained in Hoffman and Wolfe (1985), Applegate et al. (2006), and Cook (2011).
TSP is in the category of combinatorial optimization problems. If one can find a good solution in polynomial time for the TSP problem, then efficient algorithms could be found for all other problems in the NP-complete class [6].
When the salesman can get from every city to every other city directly, then the graph is said to be complete. A round-trip of the cities corresponds to some subset of the lines and is called a tour or a Hamiltonian cycle in graph theory. The length of a tour is the sum of the lengths of the lines in the round-trip [6].
A traveling salesman wants to visit each of a set of towns just once, moving from one node (as our problem is considered a graph), and returning to the same node. These nodes that should be visited once are our cities. Graph G is a complete graph [7], and the problem is to find the shortest path for this trip.
If i and j are two points from n, where n is the set of points, then d(i, j) = L (i,j). Where l stands for length and d stands for distance, the graph satisfies the triangular inequality [8]: L (i, j) < L (i, k) + L (k,

j) for the points i, j, and k. The shortest distance matrix is assumed to be symmetric [8]. When we have a fully connected graph as in TSP, and the above inequation stands as well, it guarantees to find the shortest tour for all the n points with this condition that each point should be visited just once [8]. For the TSP there is $(n-1)!$ ordering for the points and $(n-1)!/2$ solutions because each solution that contains a complete tour can be run in two directions.
It has been shown that the symmetric and asymmetric TSPs are equally difficult in the sense of NP-completeness [9]. The difficulty of TSP leads to the appearance of so many Heuristics to solve large data with minimum cost.
There are so many optimization problems that are NP-hard [10], therefore, practically solving such problems is impossible especially when it comes to big data, as when the size of the input increases, the responding time increases exponentially.

Some of the TSP problems are Euclidean (with travel metrics) and some are non-Euclidean. In the Euclidean Traveling Salesman Problem, there are n points in R^d space with Euclidean distance between any two points.

The main difference between the two models is related to the below rules and if they are satisfied, we can name our TSP problem as Euclidean TSP, otherwise non- Euclidean [8].
Rule 1: There should be no intersection to the tour itself for an optimum tour.
This rule is valid as if we have two links that are intersecting, then we can replace them with the two other links that don't violate this rule (they are not intersecting links) and their length is less than triangle inequality. When C refers to the cost that should be the minimum length in a tour, we have the below triangle inequality for all (i, j, k) [8]:

$$C_{ik} \leq C_{ij} + C_{jk} \tag{2.1.1}$$

Rule 2: For a convex hull of the points, when n is the number of the point, then the order of m points ($m \in n$) as an optimum

solution should be the same as the order of m in a convex hull. It can be interfered with rule 1, as in rule 1 we refer to the intersection [8].

These two rules reduce the number of candidate answers and consequently the number of calculations for a problem, so they are used for large-scale problems. There are different types of mathematical formulations for TSP. To formulate the Asymmetric TSP (ATSP) on m cities, one introduces zero-one variables [6], [8]:

$$x_{ij} = \begin{cases} 1 & if\ there\ is\ an\ edge\ between\ i\ and\ j\ is\ in\ the\ tour \\ 0 & otherwise \end{cases} \tag{2.1.2}$$

The subtours are for avoiding disjoint loops in the graphs. So, the below constraints are introduced to eliminate them where, K is any nonempty proper subset of the cities 1, ..., m.

$$min \sum_{j=1}^{m} \sum_{i=1}^{m} c_{ij} x_{ij} \tag{2.1.3}$$

s.t.

$$\sum_{j=1}^{m} x_{ij} = 1; for\ i = 1, ..., m \tag{2.1.4}$$

$$\sum_{i=1}^{m} x_{ij} = 1; for\ j = 1, ..., m \tag{2.1.5}$$

$$\sum_{i=K} \sum_{j=K} x_{ij} \leq |K| - 1; for\ all\ K \subset 1, ..., m \tag{2.1.6}$$

The cost c_{ij} is allowed to be different from the cost c_{ji}. Note that there are $m(m-1)$, $(0-1)$ variables in this formulation. In the symmetric TSP (undirected): $c_{ij} = c_{ji}$. To find a tour in this graph, one must select a subset of edges such that every node is contained in exactly two selected edges. In the 2-matching problem, we have

$m(m-1)/2, (0-1)$ variables. For the deletion of the possible sub-tours, some constraints are applied where $x_j \in \{0,1\}$ be the decision variable, j runs through all edges E of the undirected graph and c_j is the traveling cost of that edge:

$$min\ (\frac{1}{2}) = \sum_{j=1}^{m} \sum_{k \in J(k)} c_k x_k \tag{2.1.7}$$

s.t.

$$\sum_{k \in J(j)} x_k = 2; for\ all\ j = 1, ..., m \tag{2.1.8}$$

$$\sum_{j \in E(K)} x_j \leq |K| - 1; for\ all\ K \subset 1, ..., m \tag{2.1.9}$$

$$x_j = 0\ or\ 1; for\ all\ j \in E \tag{2.1.10}$$

where J(j) is the set of all undirected edges connected to node j and E(K) is the subset of all undirected edges connecting the cities in any proper, nonempty subset K of all cities.
In the asymmetric TSP, we have: $c_{ij} \neq c_{ji}$ for all i and j. In the asymmetric TSP, the weight of an edge is dependent on the direction but in the symmetric form of TSP, the weight is independent.
Some examples from the literature of TSP show simulation of some problems such as computer wiring, vehicle routing, clustering a data array, and job-shop scheduling [11].
The illustration of the computer wiring in the form of TSP [11] is shown in Figure 2.1.1.

It is good to explain the optimization problems and some theories. These problems are usually grouped into two categories deterministic or stochastic. According to the definitions, deterministic methods are inefficient in large-scale problems, despite the stochastic methods which are not dependent on the function properties and they can search for the global optimal answers for all types of cost functions to solve real-world complex problems.

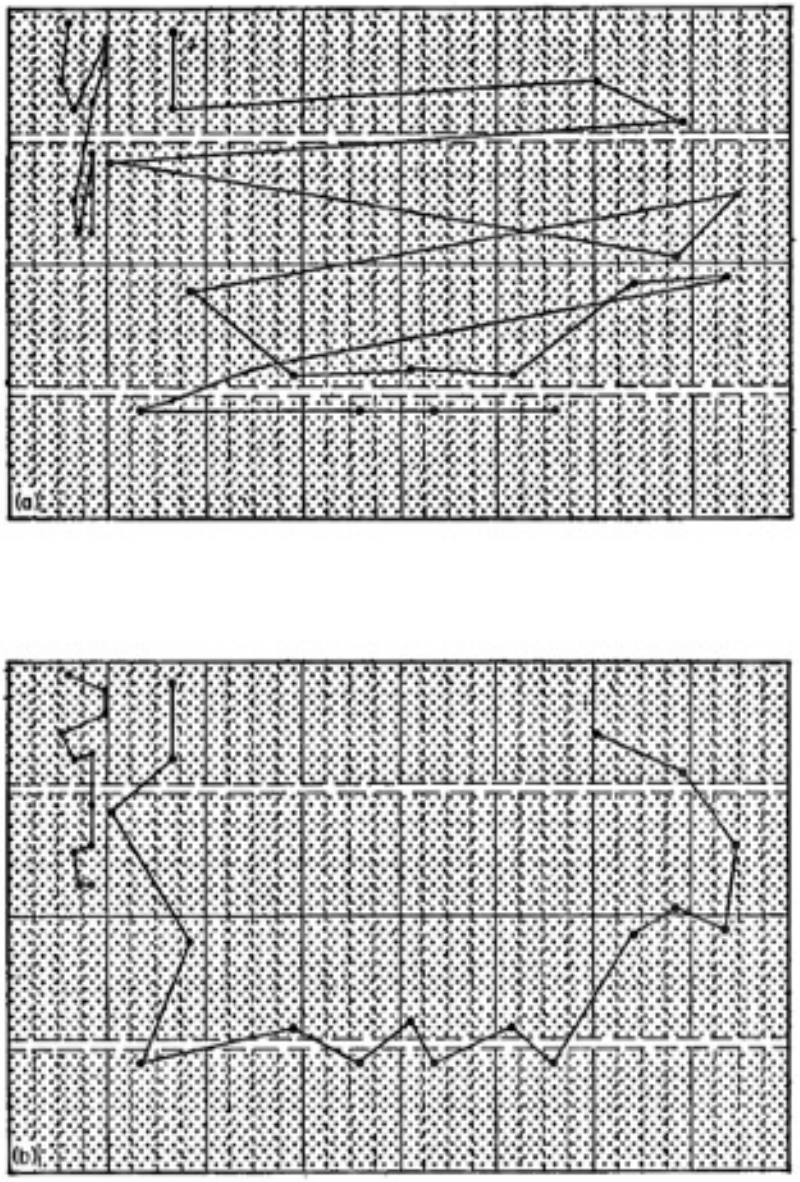


Figure 2.1.1: Figure1(a) Wiring without Optimization, Figure1(b) Wiring with Optimization (3-Optimal wiring) [11]

2.2 The Quadratic Assignment Problem (QAP)

The quadratic assignment problem is a special case of the placement problem. The problem of QAP is defined as finding the minimum allocation cost over all permutations ϕ, for a cost matrix $C = [c_{ij}]$, and the distance matrix $D = [d_{kl}]$ [12].

This NP-hard problem (by Koopmans and Beckmann in 1957) is about the assignment of a set of facilities to a set of locations (each facility for exactly one location) when the cost function is minimized and is related to the distance [13].
In this problem, costs are known and are associated with a facility being placed at a certain place (location). For the three n × n input matrices with elements such that $F = (f_{ij})$, $D = (d_{kl})$ and $B = (b_{ik})$, where f_{ij} is the flow between the facility i and facility j, d_{kl} is the distance between the location k and location l, and b_{ik} is the cost of placing facility i at location k [12].

In the original model, QAP is defined as: For n facilities and locations where N is a set (N = 1, 2,...,n) [12]:

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi_i \phi_j} + \sum_{i=1}^{n} b_i \phi_i \tag{2.2.1}$$

In this formula, S_n is the set of all permutations $\phi : N \to N$. Each product $f_{ij} d_{\phi(i)\phi(j)}$ is the cost of assigning facility i and facility j to the location $\phi(i)$ and $\phi(j)$, respectively [14]. The matrices of F and D are non-negative and symmetric. It will be depicted as QAP (F, D, B) when B is not zero. In the backboard wiring Steinberg (1961), we assume that we have some devices like controls and displace that should be connected by wire such that the total length of wires becomes minimal. If we have n devices, and d_{kl} shows the length (distances) from k to l positions, and the flow Matrix is F such that $F = (f_{ij})$, we have the next formula:

$$f_{ij} = \begin{cases} 1 & if\ device\ i\ is\ connected\ to\ device\ j, \\ 0 & otherwise \end{cases} \tag{2.2.2}$$

QAP can be used to solve problems like facility location, parallel and distributed computing, and combinatorial data analysis [15]. QAP can be used for modeling combinatorial optimization problems, such as TSP, bin-packing problems, maximal clique, and graph partitioning which can be formulated as a QAP [13]. It can be used as a test for the GRIBB project (GReat International Branch-and-Bound search).

Exact Solution An exact algorithm for a combinatorial optimization problem provides the global optimal solution to the problem. Several exact algorithms have been used for solving the QAP, like a branch and bound, and cutting plane [16].

2.3 The Open Vehicle Routing Problem (OVRP)

The open vehicle routing problem (OVRP) is an NP-hard problem that consists of finding the optimized routes for some vehicles to minimize the number of vehicles used to deliver where each vehicle moves just in one way such that the total weight (distance or time) become minimized [7]. In this case, each traveling way (route) consists of a set of customers that begins at the depot and ends at one of the customers (not specifically the first one).
The constraints of OVRP are:
1) The capacities for all the vehicles are equal;
2) There is a threshold for the traveling time;
3) The customer demands must not be greater than the capacity of each vehicle in a way;
4) Each customer should be met just once;

We can describe the differences between a VRP and OVRP. In the VRP we have a Hamilton cycle, but in OVRP each route becomes a Hamilton path, as in OVRP the vehicles do not return to the first point(depot) [25]and in case they do so, they move to the same way but in the opposite direction and via the other customers (nodes). As OVRP can be changed to an equivalent Hamiltonian cycle like VRP (not path as in OVRP), the whole problem is considered NP-hard. Any tabu search algorithm requires an initial solution from which all search processes begin [25] like the neighborhood and the consequent first move. Generally, the initial solution can be easily obtained by a meta-heuristic like nearest neighborhood heuristics (NNH) and pseudo lower bound or Insertion heuristics (IH).

The nearest neighborhood heuristics (NNH) is identical to NN, but NNH is applied to all of the problems in the case of [25]the OVRP while NN only improves each route. In this method, the last unrouted customer must enter the tour, and as there is no freedom for selection, the results don't have high quality. NNH makes some routes sequentially where we have unrouted customers near the depot, the algorithm assigns customers that don't violate any constraints (for ex. Capacity, max length). When there is no customer based on the constraints, a new route begins till all the customers are routed (visited).

2.4 Tabu Search (TS)

This algorithm is a very popular heuristic method proposed by Fred W. Glover (1986) for many combinatorial optimization problems. This algorithm is used in the operations research methods for example Pardalos and Resende (2002), Ribeiro and Hansen (2002), and Voss et al. (1999). It is widely used for large optimization problems to tackle difficult problems. TS is suitable to perform a global search for

a VRP, open VRP (OVRP), multi-trip vehicle routing and scheduling problem (MTVRSP), container loading problem (CLP) [19], job shop problem, etc. The TS has a direct and iterative approach. There are many methods of solving an optimal problem considering the distance, minimum cost function, maximum throughput, etc. TS algorithm overcomes the local minima/maxima [19]and provides optimal (not always the best, but near-optimal) solutions.

TSA is a Metaheuristic algorithm loosely connected to the evolutionary algorithm which has several versions for solving different problems, but the basic concept is the same. TS can reduce the critical areas in search space while memory can be short/long term (adaptive memory) for different problems. One of the characteristics of TS is reducing the local optima problem. The simple version of TS searches different areas for search space. In the TS algorithm, cycling back to previously visited solutions is prevented by the use of memories, called tabu lists, that record the recent history of the search [20]. The algorithm should have a movement to start a search. If we consider our problem as a graph, the algorithm considers a list of candidate solutions containing the edges that are selected. Then, the best amongst them become selected. By replacing or exchanging the current and the selected edges, the algorithm moves forward. For the new solution, we have a new list and the same swapping (static or dynamic) mechanism applied with the condition that no circle should be created in this method. The first step will be finding all the moves, selecting and recording them in the memory as a list. Selection of the best path by applying a moving mechanism, and swapping. The list of candidate solutions has a limited memory for a specific time (almost limited), as finding (searching) amongst all the possible movements is costly and time-consuming, the list should have a size by using the FIFO method [21].

There are some conditions called aspiration criteria which permit the algorithm to move in the best possible way based on the best possible discovered solution for TABU search. We have some strategies like intensification and diversification in the Tabu search algorithm where

intensification is related to modifying the selection rules specifically for intermediate memory which is a memory recently used for the initial or best-found answers with the help of some traditional algorithms. This method generates neighbors by grafting or evaluation strategies. In case we use local optima, if Tabu can not find the other interesting parts, the second method (diversification) applies to the long-term memory. These parts are not found during the intensification process. The second strategy extends the search space to different directions to find the best move, and at the same time records all the iterations till obtaining the best results. The two strategies work together, and in the case of symmetric problems, diversification is very effective. Preventive Maintenance (PM) is necessary and required because it decides the allocation of resources and tasks scheduling, memory, and data storage for the searches farther than optima points.

In the case of TSP, TSA is used successfully. A Tabu algorithm can find the optimal direction for an undirected graph (tree). In this case, the initial solution is the longest path that should be found by a greedy-based algorithm that performs a local search. TSA presented different approaches to solving a TSP like angle-based TS, parallel adaptive tabu search, and multi-point tabu search [22], [23], [24].

In the next, the flowchart and the pseudocode of the TSA algorithm are depicted in Figure 2.4.1 and Figure 2.4.2, respectively:

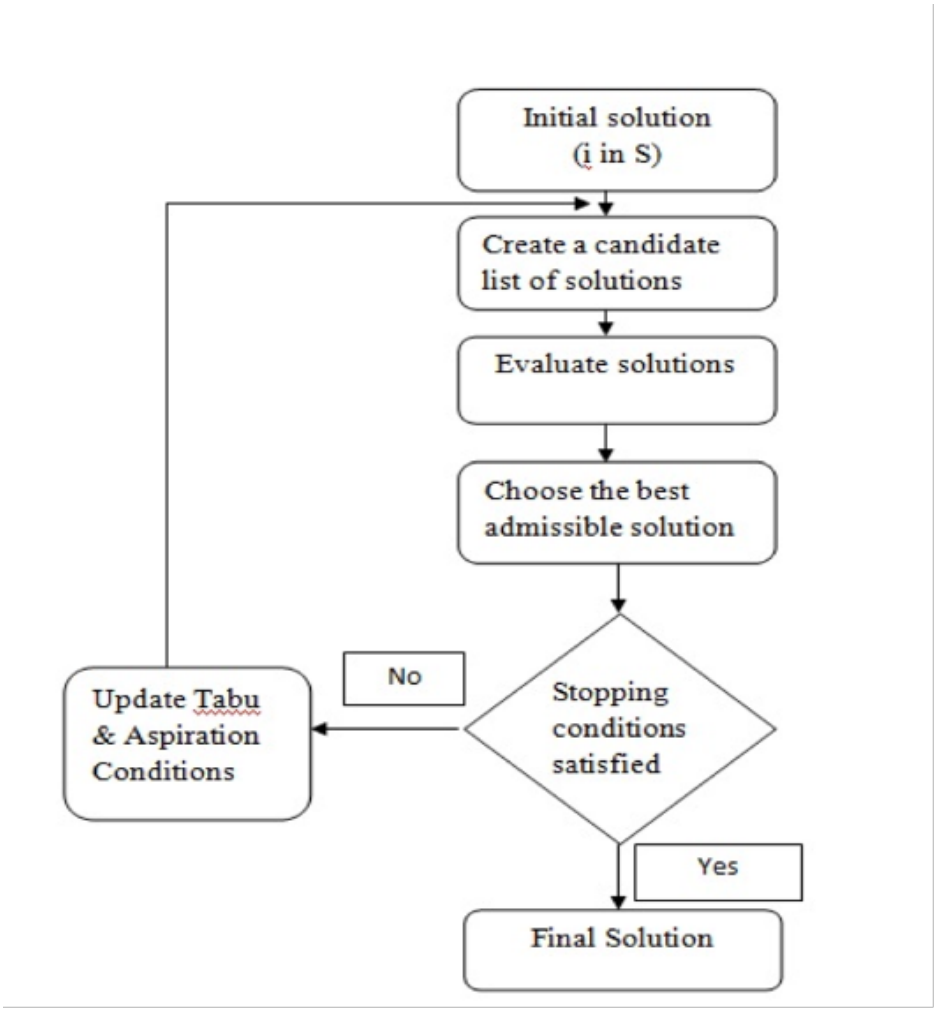


Figure 2.4.1: Flowchart of the Tabu Search Algorithm [19]

```
algorithm tabu search
begin
  tabu_list:= [];
  S:= initial solution;
  S*:= S;
  Repeat
    find the best admissible solution S1 belongs to Neighborhood of S
    if f( S1) > f(S*) then S*:= S1;
    S:= S1;
    Update Tabu list tabu_list;
  Until stopping criterion;
End;
```

Figure 2.4.2: Pseudocode of Tabu Search [19]

2.5 Test Functions

Here is the list of some mathematical functions that can be used for checking the performance of the algorithms (like 20 dimensions) [28]:

1- Sphere:

$$f_1(x) = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.5.1}$$

2- Rosenbrock,

$$f_2(x) = \sum_{i=1}^{n-1} (100(x_{i+} - x_i^2)^2 + (x_i - 1)^2) \tag{2.5.2}$$

3- Schwefel functions:

Schwefel 3.1:

$$f3.1(x) = \sum_{i=1}^{n} [-x_i \sin(\sqrt{|x_i|})] \tag{2.5.3}$$

Schwefel 3.2:

$$f3.2(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{2.5.4}$$

Schwefel 3.3:

$$f3.3(x) = max|x_i|\,, 1 \leq i \leq n \tag{2.5.5}$$

Schwefel 3.4:

$$f3.4(x) = \sum_{i=1}^{30} (\sum_{j=1}^{i} x_j)^2 \tag{2.5.6}$$

4-Griewank:

$$f_4(x) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) \tag{2.5.7}$$

5-Ackley:

$$f_5(x) = \textbf{-a}\, exp\,(-0.02\sqrt{n^{-1}\sum_{i=1}^{n} x_i^2}) - exp\,(n^{-1}\sum_{i=1}^{n} cos(2\pi x_i)) + \mathbf{a} + \epsilon, \mathbf{a} = 20 \tag{2.5.8}$$

6-Rastrigin:

$$f_6(x) = 10\mathbf{n} + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)) \tag{2.5.9}$$

7- Quartic:

$$f_7(x) = \sum_{i=1}^{n} i\chi_i^4 + \textbf{rand} \tag{2.5.10}$$

For low dimensional data, we have these functions:

8- Branin, (2 dimensions):

$$f_8(x) = a(x_2 - bx_1^2 + cx_1 - d)^2 g(1-h)cos(x_1) + g \tag{2.5.11}$$

Where,

$$a = 1, b = 1.25\pi^{-2}, c = 5\pi^{-1}, d = 6, g = 10, h = 0.125\pi^{-1}$$

9- Camel Back-6 Hump, (2- dimensions):

$$f_9(x) = 4x_1^2 - 2.1x_1^4 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{2.5.12}$$

10- de Joung (Shekel's Foxholes), (2 dimensions):

$$f_{10}(x) = (\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{j=1}^{n}(x_i + a_{ij})^2}) \tag{2.5.13}$$

where,

$$a_{1j} = -32, 0, 16, 32$$

for

$$j = 1, 2, \ldots, 5,$$

$$a_{1k} = a_{1j}$$

for,

$$k = j + 5, j + 10, j + 15, j + 20, j = 1, 2, \ldots, 5,$$

$$a_{2j} = -32, -16, 0, 16, 32$$

for

$$j = 1, 6, 11, 16, 21,$$

$$a_{2k} = a_{2j}$$

for

$$k = j + 1, j + 2, j + 3, j + 4, j = 1, 6, 11, 16, 21.$$

11- Goldstein and Price, (2 dimensions):

$$f_{11}(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 - 6x_1x_2 + 3x_2^2))\times (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)) \quad (2.5.14)$$

12- Hartmans: The constants are (c, a, and p)
For 4 dimensions:

$$f_{12.1}(x) = -\sum_{j=1}^{4} c_i \ exp(-\sum_{i=1}^{4} a_{ij}(x_j - p_{ij})^2) \quad (2.5.15)$$

for 6 dimensions:

$$f_{12.2}(x) = -\sum_{i=1}^{4} c_i \ exp(-\sum_{j=1}^{6} a_{ij}(x_j - p_{ij})^2) \quad (2.5.16)$$

13- Kowalik: The constants are (a and b)
For 4 dimensions:

$$f_{13}(x) = \sum_{i=1}^{11}(a_i - \frac{x_1(1 + x_2b_i)}{(1 + x_3b_i + x_4b_i^2)})^2 \quad (2.5.17)$$

14- Shekels: The Constants are (c and a)
all For 4 dimensions:

$$f_{14.1}(x) = -\sum_{i=1}^{5}((X - a_i)(X - a_i)^T + c_i)^{-1} \tag{2.5.18}$$

$$f_{14.2}(x) = -\sum_{i=1}^{7}((X - a_i)(X - a_i)^T + c_i)^{-1} \tag{2.5.19}$$

$$f_{14.3}(x) = -\sum_{i=1}^{10}((X - a_i)(X - a_i)^T + c_i)^{-1} \tag{2.5.20}$$

Chapter 3

Data Mining

Data mining is the process of discovering interesting patterns from massive amounts of data. As a knowledge discovery process, it typically involves data cleaning, data integration, data selection, data transformation, pattern discovery, pattern evaluation, and knowledge presentation. Based on Han, J., Kamber, et al [2]:

- Data cleaning which means deletion of noise and inconsistent data)
- Data integration (merges data from multiple sources into a coherent data store such as a data warehouse.)
- Data selection (where analytical data are retrieved from the database)
- Data transformation (e.g., normalization)
- Data mining (the process of finding hidden patterns by using some methods).
- Pattern evaluation (to distinguish the useful patterns based on some measures)
- Knowledge representation (visualization) [2].

The goal of data mining is to turn data into knowledge or information that can be used to make or develop an application for market basket analytics, fraud detection, and so on. The first four steps are called data pre-processing. we prepare data at these steps for the mining step to find frequent patterns.

Figure 3.0.1: Data Mining Techniques

Some techniques and algorithms of data mining are explained in the next subsections. Association rules, clustering, classification, regression. Figure 3.0.1 introduces some data mining techniques in the form of a tree structure.

3.1 Association rules

Association rules were popularised because of the article of Agrawal in 1993 with many citations, but this topic was introduced in a paper in 1966 by Peter Hájek et al. Let [29]:

$$I = i_1, i_2, ..., i_n$$

be a set of n items, and Let [29]:

$$D = t_1, t_2, ..., t_m$$

be a set of m transactions.

TID is a unique ID for transactions including a subset of the items. A rule is defined as an implication of the form:

$$x \rightarrow y, where\ x, y \sqsubseteq I$$

The goal of association rules is to find associations, relationships, or patterns in large-scale transactional datasets (databases). One example is market basket analytics. We can find strong rules with the help of their interestingness. For selecting all possible rules, we need to have a constraint so-called threshold that is defined for support and confidence of that rule. The rule that has a value less than the defined value (min _sup and min _conf. This model is a Feature Based Modelling framework.
In the definitions, a frequent item set is a set of items that appear together frequently where the frequency is defined by measuring the support count value. Support is the number of occurring special item(s) in a set of T transactions in dataset(D). Here is the definition of support and confidence:

Support: Means the number of appearing of an item in a dataset D.

$$Support = \frac{Supp(X \cup Y)}{N} \tag{3.1.1}$$

For example, in a transactional dataset, the number of times that these items (i1, i2, i3) are seen together.

Confidence: Means how often a rule is true:

$$Confidence = \frac{Supp(X \cup Y)}{Supp(X)} \tag{3.1.2}$$

Some famous algorithms that are introduced in this field are the Apriori Algorithm, FP-Growth, Eclat, etc.

Breadth-first search (BFS) is being used to find the shortest path in an unweighted graph G (V, E) when V and E refer to nodes and edges, respectively. It has different levels or stages such that it starts at the root of a tree in a graph, then it finds the neighbors of that node to go to the next level. Later it finds the closed nodes of the neighbors that are not visited yet, and so on. In each step, the algorithm selects unvisited nodes. The algorithm terminates when all the nodes have been visited once.

Apriori is a "bottom-up", levelwise, breadth-first algorithm [30]. Apriori was originally made to extract frequent itemsets from market basket data that are typically sparse. In these datasets, the number of frequent itemsets is small, therefore, Apriori can perform very well for them, but for dense datasets, Apriori is not too efficient. As a consequence, Apriori has been extended by other mining algorithms for example Apriori-Close, Pascal, Zart, Eclat, Charm, Eclat-Z, and Charm-MFI [30].
The steps of this algorithm for a transactional dataset (usually in market basket analysis) are:
First, we make a table and assign min_Support (for ex. 2) and min_Confidence (for ex. 50 percent). In each iteration, we have to find the frequencies of the items to find the candidates. For example: If we have 8 transactions in a table and a list of the items (for each transaction), We calculate how frequently 1-item set I_1 is found in the whole table, then we check it for the other items, for example, I_2, I_3, ..., I_n.
We make a table of all the items with their frequencies. Then we check the frequencies with the min_Supp. If it is greater or equal to the min_Supp, we keep it, otherwise, for the next step (next table) we prune it (delete it). From the new table, we calculate the appearances of the remained items two by two till we find all the frequencies

and again we compare the frequencies with the min-supp and prune the table. In the next step, for 3-itemsets, the algorithm checks the frequencies. We keep joining the items till no item remains, and we can't expand it anymore. Then from the obtained results, we find the subsets and we compute the new min_supp and min_Conf to find the valid rules. A valid rule should have a greater value than its min-supp and min_conf.

3.2 Clustering (unsupervised learning)

which is also called data segmentation, or data partition is the partition of data into some groups with similar objects. The analysis which is also called cluster analysis has a very important role in data mining. The main difference between classification and clustering is summarized such that in clustering we don't have any knowledge achieved earlier or any experience about the data, so we can partition our data without any prior knowledge to discover some information amongst the data. To do this work, similar objects must be placed close to create a cluster. The closer objects have more similarity, and vice versa.
We can consider clustering as a technique with different approaches: hierarchical-based, partitioning-based, density-based, model-based, grid-based, and soft-computing methods [43]. When k is the number of clusters, the main role of these approaches is to find clusters with minimized dissimilarity inside of the clusters (intra-cluster dissimilarity) and maximum dissimilarity between the outer clusters (between clusters dissimilarity). The above-mentioned methods help to group our data based on this. In other words, we want to find the most similar objects (based on distance) and put them in the same cluster. When we have some clusters that share the same similarity(inside) and maximum dissimilarity outside, we can find outliers as well. That is called outlier detection.

a) Partition-based methods In Partition-based or centroid-based methods of clustering, we explain k-means, k-medoids, k-median, etc.

K- means a partitioning-based algorithm that has an iterative approach that divides a given data point into K clusters. The quality of K-means clustering can be computed by the within-cluster squared error criterion. The pseudocode is [31]: **Inputs:** K: cluster numbers or the initial centroids,
D: a set of n objects,
Output: K clusters,
Method
(1) arbitrarily selection of K objects for K clusters from D as centroid,
(2) repeat,
(3) re(assignment) of the objects to the closest (most similar) clusters, Use the average(mean value) of the objects within each cluster and subtract it from each object within that cluster to find the closest one (the new assignments)
(4) Repeat the same calculations in an iterative approach for possible new assignments (calculation of the mean value for the new-found clusters again)
(5) Termination criteria: Maximum iteration number or if the shape of the clusters doesn't change.

The sum of squared error (SSE) between the objects in the clusters and the centroid c_i is measured [2] by the next equation that defines the quality of a cluster by calculation of the within-cluster variation in the form of $dis(p, c_i)^2$.

$$E = \sum_{i=1}^{k} \sum_{p \in c_i} dis(p, c_i)^2 \tag{3.2.1}$$

K-medoids: is another form of partitioned-based clustering algorithm that is sometimes called a variant of k-means. This algorithm is not sensitive to outliers or noises [32]. As for k-means, the algo-

rithm uses the mean, and the results of clustering are influenced by big values, but in k-medoids, rather than using the mean point, the algorithm uses an actual point within the cluster to represent it. A medoid is an object that is almost located at the center and has a minimum sum of distances to other points. Example: [33]PAM (Partitioning around the medoids) is the method of clustering that relies on the K-medoids. This algorithm encompasses a success complexity: $O(K(N-K)^2)I$, so it's not scalable for big data, therefore some new algorithms are introduced to resolve this problem, like Clustering LARge Applications (CLARA) and Clustering Large Applications based upon RANdomized Search (CLARANS) [33]. Here is the pseudocode of the PAM algorithm:
1- Select K representative points.
2- Repeatedly moves to better representatives.
3- calculate the quality of the resulting clustering for each pair.
4- An original representative point is replaced with a new point which causes the biggest reduction in distortion function.
5- At each iteration, the new respective medoids are formed by the set of best points for each cluster.
K-medians: is another variation of k-means, but with this difference, it calculates the median instead of the mean value for each cluster to find the centroids. This algorithm eliminates the errors for all clusters. The problem with the 1-norm is the problem of finding the most compact cluster with k centers. The k centers must be chosen in a way that they minimize the sum of the distances from each xi which is a point to the nearest ci which is a cluster center, and because of using Lloyd-style iteration, this algorithm can be a EM algorithm, that stands for Expectation and Maximization. Manhattan distance is used for distance to bring the attribute values from dataset D, so this algorithm can deal better with discrete or binary data sets.
In the Manhattan-distance formulation, the attributes may belong to different instances in the dataset; so, the median may not be in the input dataset. In the k- medoids, a medoid should be a real instance, but in the k- median, the median can be a combination of different

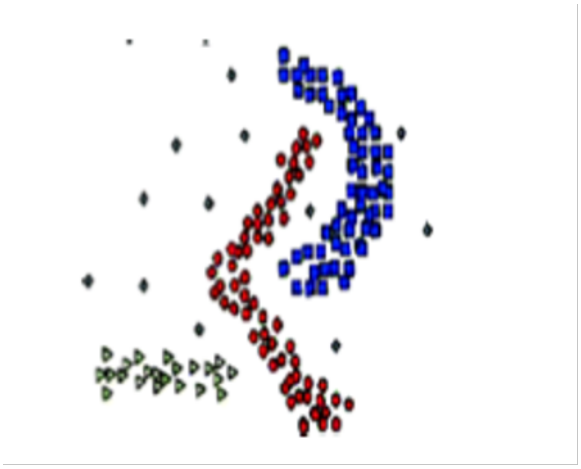


Figure 3.2.1: Example for the Density-based approach [2]

instances, because k- median uses Manhattan distance as below:
$|x_1 - x_2| - |y_1 - y_2|$

b) Density-based Method This non-parametric clustering method identifies the clusters based on their compactness in specific regions with high density compared to the other regions. This way we can find outliers in some areas with less compactness. Density-based clustering algorithms do not need to know the k value in advance and the clusters they found are not always regions with high intra-cluster similarity so they can have arbitrary shapes (natural groups). This method performs well with big data sets. In the density-based method, data points can be connected if d (distance) between them is not bigger than r (threshold), or, with direct links, a "distance" between two pairs of objects based on a definition by Hartigan (1979) is defined as below, if λ is a threshold and $p(x) > \lambda$.

$$d(x,y) = \begin{cases} -min[p(x), p(y)] & x, y \; are \; linked \\ otherwise & 0 \end{cases} \tag{3.2.2}$$

DBSCAN (Density-based spatial clustering of applications with

noise) algorithm (Ester et al., 1996) is an example that proposed using spatial index structures for a scalable clustering algorithm (see Figure 3.2.1).
For r (radius) and k (density thresholds), [2]the algorithm finds the dense regions for the points like xi if they reside in a radius (r) for the number ki of those data points when our core points are defined as $k_i > k$. Core objects O have more density (more neighbors). Two user-defined parameters called $\epsilon > 0$, ϵ (the maximum radius of a neighborhood or radius parameter), and **MinPts** (the user-defined minimum number of points needed to make a cluster) are used in this algorithm.
For the core objects, [2]the epsilon neighborhood should have at least **MinPts** objects when the epsilon neighborhood is defined as a distance within a radius ϵ centered at O. We can find all the core objects with ϵ and **MinPts**.
If this condition $d \leq r$ exists between the points, they are assumed to be connected. In DBSCAN, clusters are described as maximally connected segments of data point such that $d \leq r$, and objects that don't belong to a cluster called noise.
This algorithm is iterative such that in each iteration, it constructs new clusters and (re) assigns the data points to finally find the best clusters that cover the maximum dense regions. Points can be connected directly or transitively connected.
An object(p) [2]can be directly density-reachable from a core object (q) if $p \leq \epsilon$ - neighborhood or if p stands in the ϵ neighborhood. The equivalence relation of the density-reachability is not symmetric. The complexity of this algorithm is O (N log N), for N (the total number of points in the data set), but in the worst-case time complexity, we have $O(N^2)$.

c) Hierarchy-based methods The hierarchical methods, decompose the dataset of n objects into a hierarchy of groups which can be represented by a tree structure called a dendrogram. There are two general approaches to the hierarchical clustering method. A dendrogram has a root containing one cluster that represents all data

points. In the divisive approach(top-down), all objects are placed in one cluster which is the hierarchy's root. The root cluster divides into several smaller subclusters and recursively partitions those clusters into smaller ones. The division process continues until they reach the leaf nodes. Every node will contain only one data point. So, for n data points, we will have n leaves and n clusters, but in another method called agglomerative (bottom-up) (HAC) we have n leaf nodes (n clusters) means for each object, one cluster exists called a single cluster.
In the next steps, this method merges the data points into clusters based on their similarities (distances) until they reach the root. The root cluster must have all data points. These approaches are depicted in Figure 3.2.2 for the two forms: AGNES (for agglomerative clustering) and DIANA (for divisive clustering). A dendrogram is depicted in Figure 3.2.3.

Many agglomerative clustering algorithms have been proposed, such as single-link, complete-link, average-link, AGNES, CURE, BIRCH, ROCK, and CHAMELEON.
AGNES or AGlomerative NESting is a hierarchy-based algorithm that is not noise-sensitive and doesn't need the number of clusters in advance. This algorithm needs to be run just once. In the first step, all the objects are in individual clusters, then based on some specific criteria which can be distance measurement, they join till all of them belong to one cluster. This algorithm stands in front of the DIANA (DIvisive ANAlysis) algorithm that divides a cluster that has all the objects till each of them is placed in a single cluster.
In single-linkage clustering or nearest-neighbor clustering, the distance between two clusters is appointed by the two elements that are closest (one in each cluster). at any step, the shortest pairwise distances of the remaining clusters will be merged. The clustering result can be shown as a diagram or dendrogram.

BIRCH algorithm (Balanced Iterative Reducing and Clustering using Hierarchies), which is a hierarchy-based recursive technique that

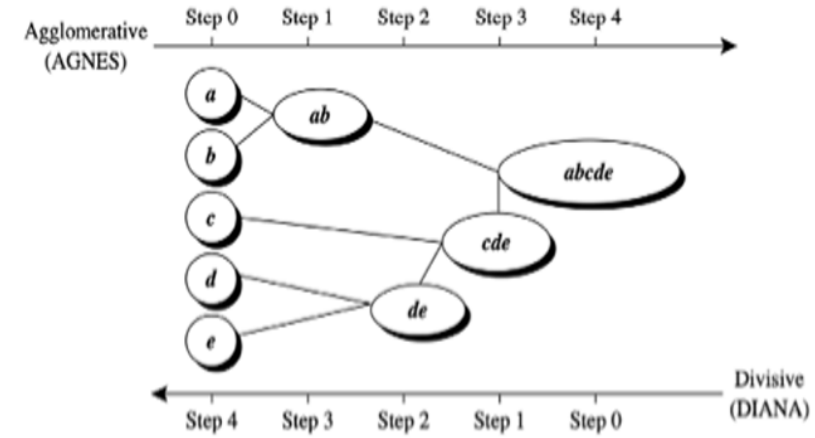


Figure 3.2.2: 1) Two approaches for five objects: AGNESS and DIANA [2]

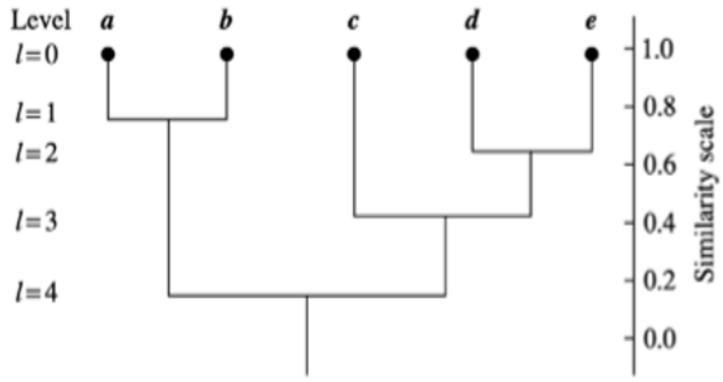


Figure 3.2.3: 2) a dendrogram representation of the mentioned objects [2]

is multi-staged and starts by inserting the data as in the pseudocode [34]. The Clustering Feature (CF) values summarize information about subclusters instead of storing all points [35]. The Branching parameter (Br) is the maximum number of children and in case of greater Br factor, the parent will be divided. The parameters are explained in Table 3.1. Here is the representation of a CF tree with its formula below and the additivity theorem for N data points [36] [37]:

CF	(Clustering Feature)
X0	(Centroid)
Radius (r)	average distance to the centroid
Diameter (D)	the average pairwise distance within a cluster
LS	Linear Sum of N
SS	Square Sum of N
Q1	mean of the radius
Q2	mean of the diameter

Table 3.1: Parameters

$$CF = (n, LS, SS) \tag{3.2.3}$$

$$x_0 = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{LS}{n} \tag{3.2.4}$$

$$R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{nSS - 2LS^2 + nLS}{n^2}} \tag{3.2.5}$$

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}} \tag{3.2.6}$$

$$CF_1 = \langle n_1, LS_1, SS_1 \rangle \tag{3.2.7}$$

$$CF_2 = \langle n_2, LS_2, SS_2 \rangle \tag{3.2.8}$$

$$CF_1 + CF_2 = \langle n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2 \rangle \tag{3.2.9}$$

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.2.10}$$

$$Q_1 = \frac{\sum_{i=1}^{k} n_i R_i^2}{\sum_{i=1}^{k} n_i} \tag{3.2.11}$$

$$Q_2 = \frac{\sum_{i=1}^{k} n_i \left(n_i - 1\right) D_i^2}{\sum_{i=1}^{k} n_i \left(n_i - 1\right)} \tag{3.2.12}$$

Pseudocode of the Birch algorithm:
1) Searching all the datasets to build a tree
2) Rebuild the CF tree with a bigger T that is the size of the tree.
3) Using one of the Clustering algorithms like with k-means, k-modes, etc.
4) Cluster refining (additional passing overs) for the new assignments based on the closeness to the centroids.
5) Repeat all the steps to create k clusters
The compactness of the clusters is defined by R, D, and CF triple [38]. The Birch algorithm uses Q1 and Q2 for the quality measurements.

3.3 Classification (Supervised learning)

is used in different fields. It is also called inductive learning and means learning from past experiences to achieve new knowledge to improve the ability to accomplish some real-world tasks. As far as computers do not have "experiences", they can learn from the collected datasets from the past, and decide based on past experiences. This way so much data with their output can represent the probable output for

another problem which can be in the field of Medicine, Engineering, etc, so it has had wide usage in recent years. Some of the famous algorithms in this field are Decision Tree, ANN (Artificial Neural Network), SVM (Support Vector Machines), DT (Decision Tree), BN (Bayesian Network Augmented), and GA (Genetic Algorithm).
Support Vector Machines (SVM) is a classification technique used for big and high dimensional data with multi-domains. In the linear version of SVM, we have a learning model and a mathematical model: $y = wx^{'} + \gamma$
If we can divide the data domain in a linear form like a straight line or hyperplane, such that the classes are separated, we call it a linear Support vector machine (LSVM), but if we can't divide it linearly and the data domain moves to a space called feature space, we call it non-linear support vector machine. The linear equation is denoted by: $y = wx^{'} + \gamma$ and the non-linear equation is denoted by: $y = w\phi(x') + \gamma$.The non-linear SVM uses a kernel function to map data.

3.4 Regression techniques:

can be adapted for prediction. In regression, we have one or more independent variables known as attributes or features (X) and dependent variables known as response variables for outcomes (Y) that we want to predict. In this model, the dependent attribute model should be predicted by the attribute variables. Regression has two types: LR (Linear Regression), and NLR (Non-Linear Regression).

$$Y = F(X) \tag{3.4.1}$$

3.5 Grid-based Techniques:

In the CELL method, for outlier detection, our space is divided into a multidimensional grid, such that each cell becomes a hypercube with

a diagonal of length [2] r/2 (r is a threshold to find the neighbors), l is the dimension, and the length of each edge of a cell is $\frac{r}{(2\sqrt{l})}$.
In 2-dimensional data, we have these properties:
pruning for the first level if x and y are points, and C refers to a cell, for in a level -1 cell, we have: $d(x, y) \leq r$, otherwise, for a level $-$ 2 cell pruning, we have: $d(x, y) \geq r$. After grouping the data (if groups belong to one of these rules), we can find the outliers, then there is no need to check all the data (the objects in each group), and just the cells that are closer to our goal (that is our target cell) will be checked. The examples are STINGS and CLIQUES.
CLIQUES (Clustering In QUEst) algorithm is a subspace clustering method like the Apriori algorithm. CLIQUE is a kind of grid-based clustering algorithm that looks for high-density region areas in subspace. The method works by partitioning the dimensions into some intervals to partition all the space into some cells by using a threshold for finding the dense regions and noises or sparse cells. If the number of objects transferred to the cell becomes more than that threshold, that part is called dense. Clique works like the Apriori algorithm in association rules for frequent pattern mining. By using the monotonicity of the denser cells, it finds a candidate solution that is a candidate search space. If k stands for dimensions and$k \geq l$, c for cells, l points threshold, then a k dimensional c has minimum l points I if k-1 dimension projection of c has minimum l points. The steps of cliques are:
1) Partitioning of d-dimensional data into some parts(units), finding the dense regions by locating the intervals with minimum l points, then joining 2-dimensional dense areas in the subspace, repeatedly. Clique checks the threshold for the new dimension k+1 till no candidate solution or dense region remains.
2) Assembling the clusters greedily such that every cell is placed in a dense region and makes a cluster with an arbitrary shape. In this phase, the clique uses MDL (Minimum Description Length) to apply maximal regions (a hyperrectangular shape that holds dense cells and is unextendible to the higher regions) to cover the dense areas that

are already connected.
Therefore, the CLIQUE algorithm begins with [2]:
a) an arbitrary dense cell,
b) finding a maximal part that covers the cell, and
c) working on the uncovered dense cells till covering all the cells.

Advantages
1)Automatically find subspaces with the most dense clusters.
2) It is not sensitive to the input orders and any canonical data distribution.
3) linearly scalable for high-dimensional data
Disadvantages
1) Setting the grid size (a constant structure) and the density threshold l for all the dimensions can affect finding good clusters.
2) All projections to a lower dimensional space can be dense and become overlapped amongst the dense regions.
3) Finding some clusters with different densities in different subspaces is not easy.

Chapter 4

Metaheuristics

Metaheuristics have been used for solving several optimization problems as they are more powerful than conventional methods and to cope with large data. Intensification and diversification are two main features of the metaheuristic algorithms. The intensification phase searches around the current best solutions and selects the best candidates or solutions. The diversification phase ensures that the algorithm explores the search space more efficiently. The specific objectives of developing modern metaheuristic algorithms are to solve problems faster, to solve large problems, and to obtain more robust methods. In these algorithms, nature is considered as an inspiration. Biologically inspired algorithms are one of the main categories of nature-inspired metaheuristic algorithms. The bio-inspired algorithms mimic the best features in nature. They behave based on the selection of the fittest in biology which have been derived through the evolution by natural selection over so many years in the history of the world. In recent years, many of the algorithms that are developed based on them have become the main topic of research. We can place these algorithms in three groups which are defined in some categories [28]:
(1) Evolutionary algorithms (EA),
(2) Swarm intelligence (SI),
(3) Bacterial foraging algorithms.

The first group is generally genetic-based algorithms. Evolutionary Algorithms (EAs) are some methods that are called heuristics based on the evolutionary rules of genetics discovered by Darwinians. The purpose of these algorithms is to find the global answer (best answer or near optimum) for problems that are highly probable to be discovered sooner in the first iterations. Some of the well-known paradigms of EAs are Genetic algorithm (GA), genetic programming (GP), evolutionary strategy (ES), and differential evolution (DE) which are population-based with stochastic search algorithms considering the criteria (best to survive). Some of the sources which a metaheuristic algorithm can be inspired from are depicted in Figure 4.0.1.

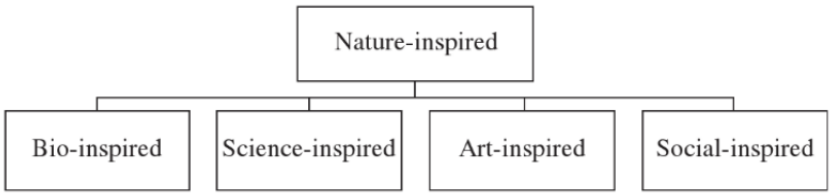


Figure 4.0.1: Sources of Inspiration for the Metaheuristic Optimization Algorithms [39]

In other definitions, Meta-heuristic algorithms are grouped as:
1) evolution-based: Like a Genetic algorithm that improves during the evolutions and always finds the best answer and combines them.
2) physics-based: Based on the physical rules in the world, like SA (simulated annealing)
3) Swarm-based methods: Based on the group behavior of animals: Like Particle Swarm Optimization (PSO) by Kennedy and Eberhart 1995, Artificial Bee Colony (ABC) by Dorigo et al. 2006, Ant Colony Optimization (ACO) 2006, etc.
4) Inspired by human behaviors like Tabu Search (TS), ICA (Imperialistic Competitive Algorithm), Cuckoo Search (CS), League Championship Algorithm (LCA), etc.

Figure 4.0.2 shows some optimization techniques used for metaheuristics.

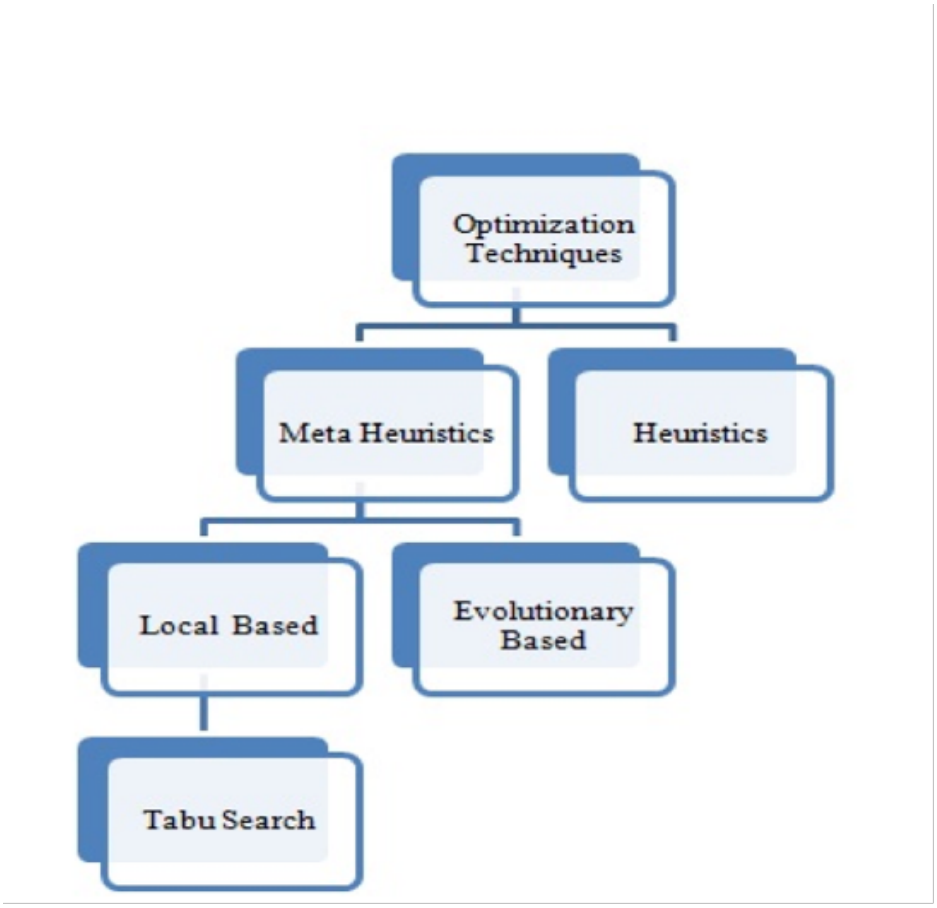


Figure 4.0.2: Optimization techniques for mathematics [19]

4.1 Genetic Algorithm

Genetic algorithm (GA) is one of the first EA algorithms (adaptive heuristic search algorithm) that performs a stochastic search (random

search). It was first inspired by the theory of evolution (natural selection) of Charles Darwin discovered by John Holland et al in the 1960s and 1970s (Holland, 1975; De Jong, 1975) [40]. Darwin's theory was based on the survival of the fittest to refine the solutions [41]. GA is an abstraction of biology that uses some operators like crossover, mutation, and parent selection to solve a problem [41]. Based on Darwin's theory of evolution, evolution happens if and only if three conditions are met [42], [43]: 1) Struggle for existence 2) Variations 3) Inheritance of the variations If for any reason any one condition does not meet in the regeneration process, natural selection will stop at that generation. Table 4.1.1 defines some of the used terms in Genetics. According to Hari Mohan Pandey et al [43], Genetic behavior can be affected by the factors depicted in Figure 4.1.2.

Chromosome	A string of DNA in the cell nucleus that carries the genes
Crossover or recombination	Transferring genetic information which are taken from two parents to produce offsprings.
Mutation	A small random change of a genotype
Genotype	The collective genetic build an organ (sometimes called chromosome)
population	A group of solutions that are searching in parallel.
Fitness	A goodness measurement
Phenotype	The representation of a genotype achieved by decoding of genes

Figure 4.1.1: Some of the genetic terminologies

The basic processes of Genetics:
1) Selection of the individuals for the next generation
2) Manipulation of the selected individuals to make a new generation

using the two operators [44]. Crossover and mutation search in the space, and selection reduces the search area by eliminating some of the solutions based on the selection strategy within the population. GA which is a population-based algorithm simulates the genes considering the survival probability of the fittest individual, where fitness is the cost function and the fittest individuals have more probability to pass their good genes to the next generations. In GA, chromosomes are depicted as a set of solutions and genes defined by a parameter. The new generation inherits the new genetic features in the form of offspring.
The GA algorithm selects the fittest (the best solutions) using the selection operator (For example, the roulette wheel method). The algorithm keeps the best answers of each generation to improve the other answers. Between the individuals, the crossover operand will lead to finding the place between the two parents. The genes in the chromosomes change randomly during the mutation phase.

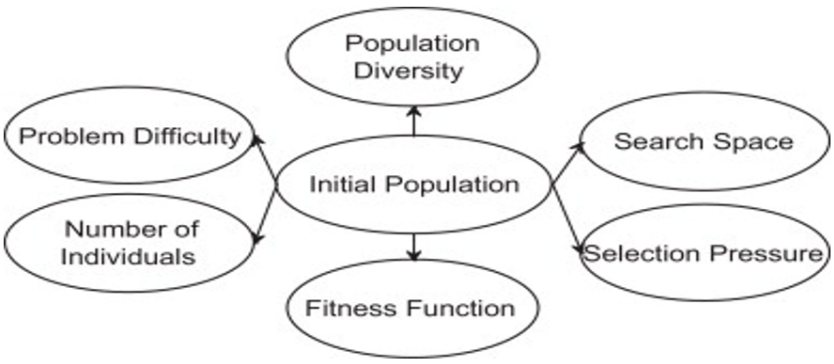


Figure 4.1.2: The affecting factors for the behavior in Genetics [43]

The GA algorithm can be used for both maximize/minimize functions and has so many advantages that they can be used in parallel, for solving complex, discrete problems (like TSP) vs continuous, graph

coloring, pattern recognition, data mining, natural language processing, multi-objective problems, and many more optimization problems, whether their cost is stationary or non-stationary (change with time). In GA, the population can search over all the space simultaneously (even in different directions) as far as the offspring of a population are considered independent. GA is widely used for non-linear problems. The disadvantages of the GA are deciding about the fitness function (the formulation), deciding about the size of the population, the ratio of operand parameters like mutation, and crossover, and deciding about the criteria of the new population (in the phase of selection). All of these choices must be done precisely, as it leads to poor solutions.

4.1.1 Selection Techniques

Selection takes place in [43]:
1) The beginning called the Initial Population selection or generation (random initial population)
2) The selection of the population for the next generation.

In GA, the process of selection is important because it has a critical role in the convergence of the algorithm if the selection method is not suitable, it can deviate the results and lead to (usually) premature convergence (because of the lack of diversity) which means it has an effect at the speed of evolution usually called selection pressure. So, the overall performance can be influenced. in the flowchart [44], there is a simple version of GA in the form of a Flowchart (Figure 4.1.3). In GA, we have some resources and some competition to achieve these resources. Any individual tries to win this competition and the most successful one will produce offspring. Genes transmit to the next generation (according to the literature) based on the selection procedure to make better offspring [45] so at the final result, we should have the best (the most fitted and dominated) results from the individuals in each population.[46] The efficiency of GA can be

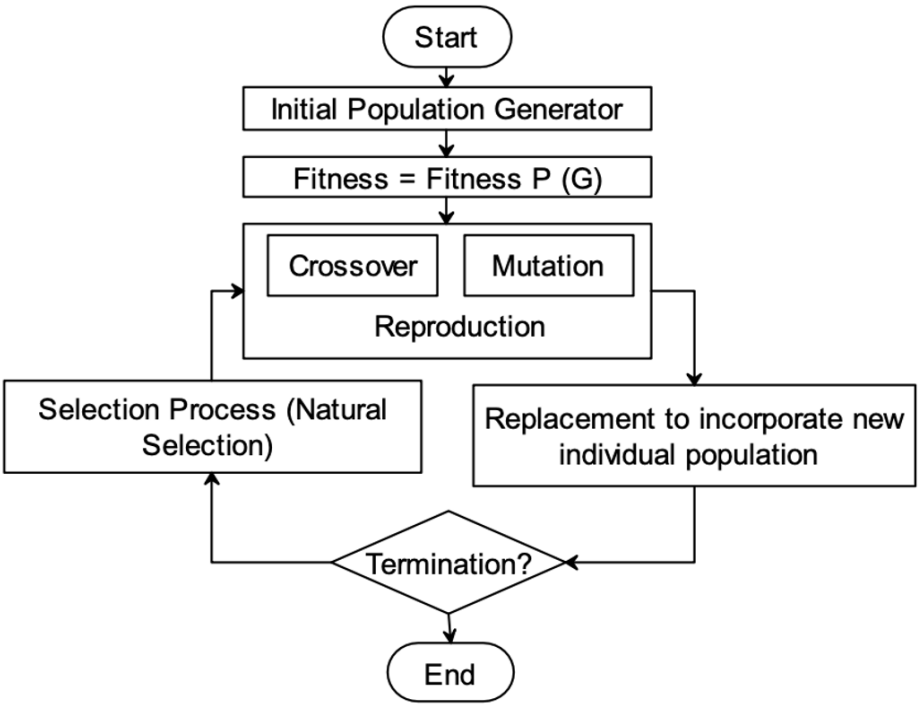


Figure 4.1.3: Simple Genetic algorithm flowchart
[44]

described by the convergence time and the number of generations for an optimal solution.
In Figure 4.1.3, when the reproduction of the crossover and mutation ends, we will have the selection of the pairs step. GA comprises a population of strings of characters in each generation that are similar to the chromosomes, where every chromosome can be a solution [47], [41].

In the EAs, the diversity of search reduces after an early convergence of the algorithm in a local optimal. It means the results of having a genotype (non-optimal) that is taken over a population are either individuals become identical or, the low genetic diversity for

later evolution.
Diversity matters because the GA wants to perform a global search and avoid premature convergence. Reaching the global optimum and genetic diversity have a key factor called selective pressure which means the selection of the best member (solution) of the current generation. If this factor is high, the diversity reduces, and in that generation early convergence happens, so this factor should always be balanced.
The selection approach is so critical because according to the selection of the most- fittest chromosomes during a global search, the next generations should have higher fitness so that the best members become selected for mutation to create the next generation.
This phase of the GA algorithm determines whether the individuals are selected or not for reproduction and also the number of seeds generated by every selected individual. The good individuals have a higher chance of being selected as parents in other words, in this phase, it will be decided which solution should stay to reproduce when the size of the population is considered as a constant). The genetic algorithm uses fitness as one of the factors for selection but there are some other measurements (strategies) for selection as sometimes relying on one factor like fitness will lead to the selection of the best answers which are not always optimal so considering the unfit individuals is necessary as well.
Therefore, we use some other selection techniques such as Proportionate Roulette wheel selection, Linear Ranking Selection, Exponential Ranking Selection, and Tournament Selection which are discussed.
1) Proportionate Roulette Wheel Selection In this approach, the probability of selection is shown as a spinning roulette wheel and Based on the schema theorem by Holland the selection of chromosomes is proportional to the probability related to the fitness of that chromosome. When we have a population partitioned on a wheel where each sector represents an individual, we can calculate the selection of an individual with the below equation that describes the proportion of the fitness for an individual (for ex. f1, f2,..., fn) to the total fitness

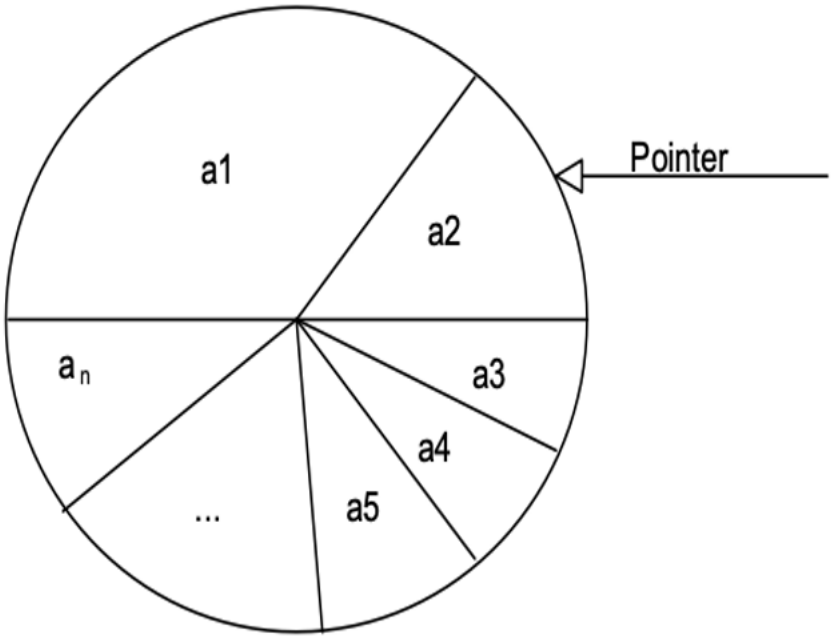


Figure 4.1.4: The Roulette wheel selection method
[46]

value of the population. So, the area of the wheel selected for an individual will be assigned. The circumstance of the roulette wheel is the sum of all fitness values. The section size is proportional to the fitness (the higher the fitness, the greater the segment size), so the higher probability belongs to the fittest one with a bigger segment, and the smallest segment has the lowest probability amongst the other chromosomes. Figure 4.1.4 shows the roulette wheel selection approach.

$$ps(a_i) = \frac{f(a_i)}{\sum_{i=1}^{n} f(a_j)};\ j = 1, ..., n \tag{4.1.1}$$

The steps for the Roulette Wheel selection method:

1) Calculation of the sum of fitness values of all individuals of a population
2) Calculation of the fitness of each individual (chromosome) that is $f(a_i)$ and their probability of selection using the above-mentioned equation
3) Partitioning the wheel into some sectors regarding the probability.
4) Spinning for 'n' times (n is the population size) till it stops and shows a sector (an individual) to be selected for the next generation.
One of the disadvantages of the Roulette wheel approach is premature convergence and a loss of diversity. When we have a population that contains very fit individuals especially when our algorithm is in the initial phase, then they will be selected very soon despite the other individuals that possibly have better fitness which the best fittest chromosomes may be discarded, and exploration stops, and sometimes the probability of selection of the fitted and unfitted individuals are the same because of having the same (similar) fitness value and it makes the exploration difficult. This approach is also difficult for solving minimization problems so the problem should become a maximization problem to be solved (like TSP).
2) Linear Ranking Selection In this approach which was first introduced by Baker (1985) [46], the steps are as:
1) Sorting individuals based on their fitness and ranking them. For the Best sorting algorithm, we have O (n log n) time complexity.
2) Assignment of 'N' for the best individual, and '1' for the worst one.
3) Applying the probability of selection P_i (for the i_{th} individual) linearly based on the ranking, where n^-/N defines the probability for the worst but n^+/N stands for the best individual according to the below equation:

$$p_i = \frac{1}{N}(n^- + (n^+ + n^-)\frac{i-1}{N-1});\ i = 1, ..., N \qquad (4.1.2)$$

The ranking of each individual is different even if they have the same ranks.

For the Ranking selection, the time complexity is O (n log n) + time of selection which can be between O(n) and $O(n^2)$.

3)Exponential Ranking Selection In this approach, the probabilities have exponential weights. The base of the exponent is c, where $0 < c < 1$. This approach and the previous approach (linear Ranking Selection) are the same but for the calculation of Selection Probability we apply the below equation, where $\sum_{j=1}^{N} c^{N-j}$ used for normalization that the summation of all p_i becomes equal to 1.

$$p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}};\ i \in \{1, ..., N\} \tag{4.1.3}$$

4) Tournament Selection Approach

This approach which is more famous and efficient than the three other approaches has the following steps:

1) Random selection of n individuals for a tournament size usually 2 for the Binary. Where the tournament size (TS) is defined as the number of individuals exist in each tournament (when the size increases, the diversity loss probability increases).

2) During a competition between the individuals, the winner will become selected which is described as an individual with the highest fitness value.

This random selection can lead to a loss of diversity because maybe some of the individuals didn't have a chance to be selected as the tournament begins with a random selection, or they didn't get selected in the intermediate as maybe the individuals lost some tournaments.

The advantages of this approach are:

1) good time complexity for example O(n),

2) easy parallel implementation,

3) low vulnerability to takeover by dominant individuals,

4) No need for scaling or sorting of fitness.

As an example, the process of Tournament Selection is shown in Figure 4.1.5, where 3 chromosomes are selected:

Gandomi and Alavi proposed multi-stage genetic programming as advanced Genetic programming for non-linear system modeling

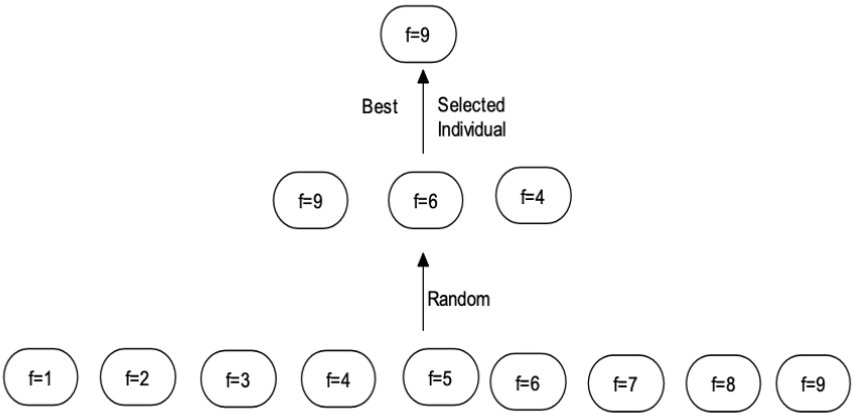


Figure 4.1.5: Tournament Selection Approach [46]

that is based on incorporating the sole effect of predictor variables with the interactions between the variables to provide more accurate simulations [28].

4.1.2 Genetic for the Traveling Salesman Problem

TSP is conceptually simple but it is difficult to obtain an optimal solution as the number of tours in this problem can be (n- 1)!/2 for symmetric n cities and when the number of cities increases, the number of permutations of valid tours are also increasing, so we can say that It is the factorial growth that makes the problem immensely difficult. For the TSP problem, the path (e.g. the order of the cities) is a usual representation of the problem where chromosomes are solutions [44], [49].
The steps for applying the Genetic algorithm for TSP can be viewed below:
1) Provide some necessary information (set the city's locations, the maximum number of generations, the size of the population, and

probabilities for the operators (crossover and mutation).
2) Creation of an initial population of individuals (chromosomes) and calculation of their fitness.
3) Creating the new population using the three operators.
4) Selection of the parents from the new population for reproduction of a child using the two operators. This is called the selection approach.
5) Evolution of the new set of chromosomes after each generation (where they have a similar size as the population size number).
6) Repeating this procedure until the convergence to the optimal value (the best path), where the majority of the population has the same optimal solutions (chromosomes).
According to Noraini Mohd Razali et al, if the probability is not proportional to the fitness, TSP can be solved. There exist two types of such non-proportional selection operators: tournament, and rank-based techniques. In the Rank-based Roulette Wheel Selection we assume the probability is related to the fitness ranking. The steps are as below:
1)Sorting the chromosomes based on their fitness
2) Calculation of the probability for the selection step which is related to the ranking (linear or non-linear) and consequently a scaling for the population
3) Mapping the indices to the list using a mapping function
4) assigning sampling method using a roulette wheel to populate the mating pool.
For n individuals, where pos indicates the position, pos= 1 is the worst fitness, and pos=n shows the best fitness for an individual. We have the below equation for linearly-scaled ranking where SP indicates the selective pressure:

$$Rank(pos) = 2 - SP + \left(2.(SP-1).\frac{(pos-1)}{(N-1)}\right) \qquad (4.1.4)$$

SP helps to control the bias where $2.0 \geq SP \geq 1.0$ and the expected sampling rate of the best individual is SP, the expected sampling rate of the worst individual is 2-SP and the selective pressure of the other

individuals can be understood by linear interpolation of the SP (based on the rank) [50]. Based on De Jong's guideline, the algorithm can begin with high PC (Probability of Crossover), low Pm (Probability of mutation), and moderated population size. The main advantage of using this method is avoiding premature convergence. Figure 4.1.6 shows how Genetics can be applied to TSP.

Algorithm-1: Procedure GA-TSP (No. Of cities)

```
Begin
Initialize GA and TSP parameters:
No. Of cities
Cities' coordinates
G_max shows the maximum number of generations
Size of the population
Crossover rate
Mutation rate
Tournament size
Generate random,  initial population P (G)
Fitness◄— Evaluate P (G)
While (((Result is not Optimum) OR (Generation< G_max))
    Do select a couple of parent population P1 from P (G)
        Apply crossover to P1
        Apply mutation to P1
        G=G+1
        Update Population (P(G),P1(G))
End while
Display optimum result
End
```

Figure 4.1.6: Procedure of Genetic for the Traveling Salesman Problem [51]

The higher the number of populations, the better the quality of the results becomes (finding minimum cost), the slower convergence, and the Higher iteration time. If the solution quality of a TSP matters more than the time specifically then the rank-based selection approach is better, otherwise tournament can give us good results in a short time as well.

In a different study (2016) [52], we have an advanced method of the Roulette Wheel approach for TSP, and based on that, Selection is the major carrier for the simulations of life reproduction which indicates the process of selecting the most fitted individuals in the

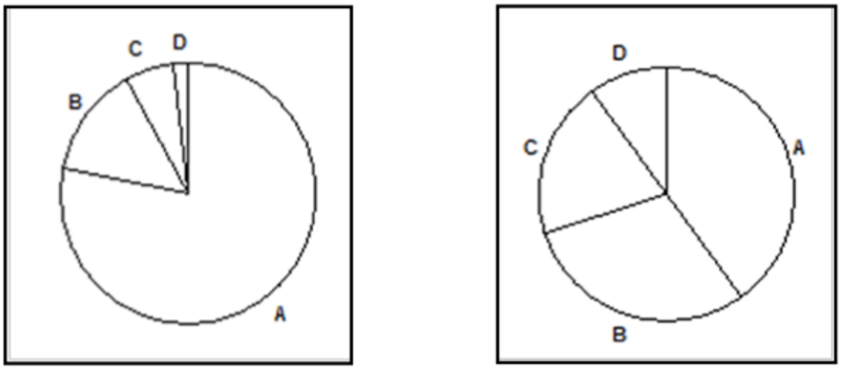


Figure 4.1.7: a) proportionate fitness; b) Rank-based fitness [46]

current group to generate a mating pool.
There are different forms of selection, including fitness proportionate selection, Boltzmann selection, elitist selection, etc [53], [54].
Fitness Proportionate Selection Mode and Optimization Method For the Fitness Proportionate Selection Mode, the fitness values of all individuals should be calculated first. Then, the proportion of the fitness in the total group fitness should be calculated, representing the probability of the individual being selected during the selection process. For a group with a given scale of n, $p = a1, a2, ..., an$ and the fitness of the individual is $f(a_i)$, $a_i \in p$, we can calculate the selection probability of individual fitness by the formula [52].

$$ps(a_i) = \frac{f(a_i)}{\sum_{i=1}^{n} f(a_j)};\ j = 1, 2, ..., n \tag{4.1.5}$$

The above formula determines probability distributions of individuals from the offspring population, among which the expected individual survival amount in the population of the parent generation is [52]:

$$p(a_i) = np;\ j = 1, 2, ..., n$$

The global idea of this selection mode is to make individuals with high fitness breed more and make individuals with low fitness breed less and even no breed. Generally, the scale of individuals from the parent generation equals that from the offspring generation. $P(a_i)$, is generally not an integer. If rounding-off is adopted, the population scale is usually changed after the summation of individuals from the offspring population. The roulette wheel method is then for this issue [52]. This model is depicted in Figure 4.1.7 [46].
They used a self-adapting crossover and self-adapting mutation operator as a new mechanism. The method kept the optimal results after applying crosses and mutations in every generation [52]. With the improved mechanism, the impacts of different selection modes on GA could be better reflected. The flowchart is depicted in Figure 4.1.8. The Genitor algorithm, for example, can also be seen as a variant of

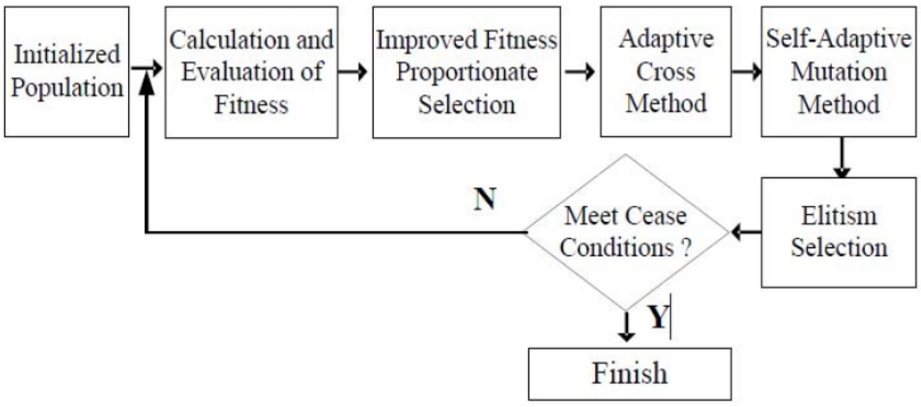


Figure 4.1.8: Flowchart of the Genetic Algorithm
[52]

a $(\mu + 1)$ EA because of its selection strategy, but in the regular form of GA, we have (μ, λ) EA where $\mu = \lambda$.
In this algorithm, after the selection of two parents, reproduction happens for an individual. The seed is immediately placed back into

the population but the worst will become deleted so the rest of the population improves monotonically. In Genitor, fitness is assigned to an individual based on rank, and the population remains sorted so the position of individuals matters for the assignment of fitness. As a result, duplicates don't happen and the best answers are always kept in a population. The Genitor algorithm has high selection pressure.
Another type of Genetic is called SGA (Stud Genetic Algorithm). In this algorithm, the selection is not stochastic, but at each generation, the fittest value (the best individual) will be selected for crossover, and the information will be shared by using the standard genetic operators. Despite of efficiency and accuracy, this algorithm can cope with so many functions in a diverse environment of a big population to find the global optima for a multi-modal search space. Some researchers like Gandomi and Alavi proposed a new version of the genetic algorithm for a non-linear system called multi-stage genetic programming which uses the unique predictor variables and their connections for simulation. Another GA-based algorithm is BBO(biogeography-based optimization) discovered by Simon is applied for the global recommendation, and uniform crossover.
The Evolutionary algorithms which do not need any information about the fitness gradient have easy parallel processing and can deal with the problem of getting stuck in local optima where the methods are not applicable or fail. EA convergence is not dependent on the target functions (continuity or differentiability). They can be used to improve performance or combined with hybrid algorithms. EAs can be used for multi-objective problems as they can work simultaneously with different variables like integer, discrete, and discontinuous design variables; accompany game strategy, they are not sensitive to Pareto front shapes and can find answers in non-convex or discontinuous locations. EAs are good for stochastic problems, uncertainties, or fitness with noise, and they have good competence as the cost of EA grows linearly with the size of the problem. EAs – specifically Genetic - are easy to execute in parallel because they are population-based search methods. PEAs (Parallel Evolutionary Algorithms) (PEAs)

which are complex non-linear algorithms are a kind of Evolutionary algorithm. Biogeography-based optimization (BBO) is another kind of EA algorithm proposed by Simon used for global recommendation and uniform crossover.

4.2 Some of the Swarm Intelligence (SI) Algorithms

According to Christian Blum et al, swarm intelligence (SI) is biology and by now it has helped a lot to solve problems by transferring the information of the members of a nest, colony, etc. These algorithms are based on the simulation of the collective behavior of animals [28]. The individuals in a group have a special behavior called collective behavior and we can solve many real-world problems by modeling these behaviors in the form of mathematical formulas. Using the decision patterns of every individual (agent) helps the computer scientist discover new solutions in a more optimized way. So many algorithms have been discovered based on the swarm collective behaviors. Each of the algorithms has been inspired by a new feature of the agents. Calculation of distance in these problems is so important because these problems are usually optimization problems and finding the best answer (path) that is more optimized has a direct connection with distance, so many distance measurements have been used to solve minimization and maximization problems. In this section, first, we want to define Particle Swarm Optimization (PSO). We explain Ant Colony Optimization, Flora Algorithm (FA), Krill Herd Algorithm (KHA), Firefly Algorithm (FA), Whale Optimization Algorithm (WOA), cuckoo search (CS) algorithm [55], [56], spotted hyena optimizer (SHO) [57], etc. Several extensions to the major categories of swarm algorithms have been presented in the literature [28], as they can be widely used in optimization. The bacterial foraging behavior is also a bio-inspired optimization approach, and the most famous types of bacterial foraging algorithms are computing systems of microbial

interactions and communications (COSMIC) and rule-based bacterial modeling (RUBAM).

4.2.1 Particle Swarm Optimization (PSO)

The background of PSO goes back to 1987 when the simplest version of PSO namely the Boid-oid model introduced by Reynolds. It was simulating the social behavior of birds as agents so later, It became an inspiration for developing PSO. In the Boid-oid model, we suppose a cartesian system where each bird is depicted as a point with a randomly selected location and velocity. After the execution, with "the nearest proximity velocity match rule," every individual [58] must have the same velocity at its closest neighbor, and after some iterations, all the points (birds) speed becomes equal. Because of the application of this model for real-world problems, they defined a variable for the speed randomly for every iteration of the program.
Heppner made a cornfield model for the simulation of the bird's behavior in 2002 (Clerc and Kennedy). Based on that model which is about position and randomly selection of birds, we must have some rules for the bird's movements:
1) Following this assumption the position coordinate of the cornfield is defined as (x_0, y_0), and (x, y) position coordinate of birds and (v_x, v_y) velocity coordinate of individual birds.
2) The performance of the current location and velocity is related to the Distance between the current location and cornfield such that the further the distance the better the performance and vice versa.

3) Assumption of the bird memories that makes them keep the information of the best position (pbest), and a (constant for adjusting the velocity) where rand [0,1], we have the below rules for every change in the speed [58]:
If $x > pbestx$, $v_x = v_x$ - $rand \times a$, otherwise,$v_x = v_x + rand \times a$.
if $y > pbesty$, $v_y = v_y$ - $rand \times a$, otherwise, $v_y = v_y + rand \times a$.

Based on the computer simulations for the two velocity adjusting constants (a and b), if the result of $\frac{a}{b}$ becomes a big number, the birds collect in the cornfield sooner (the best answer), and vice versa, so the best answer can be found sooner. Kennedy and Eberhart were inspired by this model and summarized each individual as a particle considering the speed and location and they called it PSO. They offered the below equation:

$$v_x = v_x + 2 \times rand * (pbestx - x) + 2 \times rand \times (gbestx - x) \quad (4.2.1)$$

$x = x + v_x$;
In the PSO algorithm, each individual (particle) is a solution. PSO is defined by searching which memorizes location and speed. The combination of information in each generation helps the algorithm adjust the speed of every dimension so that the algorithm can find the new positions until it finds the balanced state. The flowchart of PSO is depicted in Figure 4.2.1.

We can define PSO as a population-based stochastic algorithm inspired by the social behavior of bird flocking or fish schooling [28] presented in 1995 and later modified and changed and so many variants have been discovered by now to be used in different engineering fields. PSO has global and local versions.
Original particle swarm optimization Mathematical definition for the initial (original) [58] particle swarm optimization for swam size N, and D dimensional, where Position vectors are defined as $Xi = (xi1, xi2, ..., xi_d, ..., xiD)$, velocity as $V_i = (v_{i1}, v_{i2}, ..., v_{id}, ..., v_{iD})$, individual's optimal position $P_i = (p_{i1}, p_{i2}, ..., p_{id}, ..., p_{iD}$, swarm's optimal position as $P_g = (p_{g1}, p_{g2}, ..., p_{gd}, ..., p_{gD}$ update formula of the individual's optimal position for a minimized problem [58].

$$p^d_{i,t+1} = \begin{cases} x^d_{i,t+1} & if\ f(x^d_{i,t+1} < f(p_{i,t}) \\ p^d_{i,t} & Otherwise \end{cases} \quad (4.2.2)$$

And the Update formula of velocity and position is denoted as follows [58]:

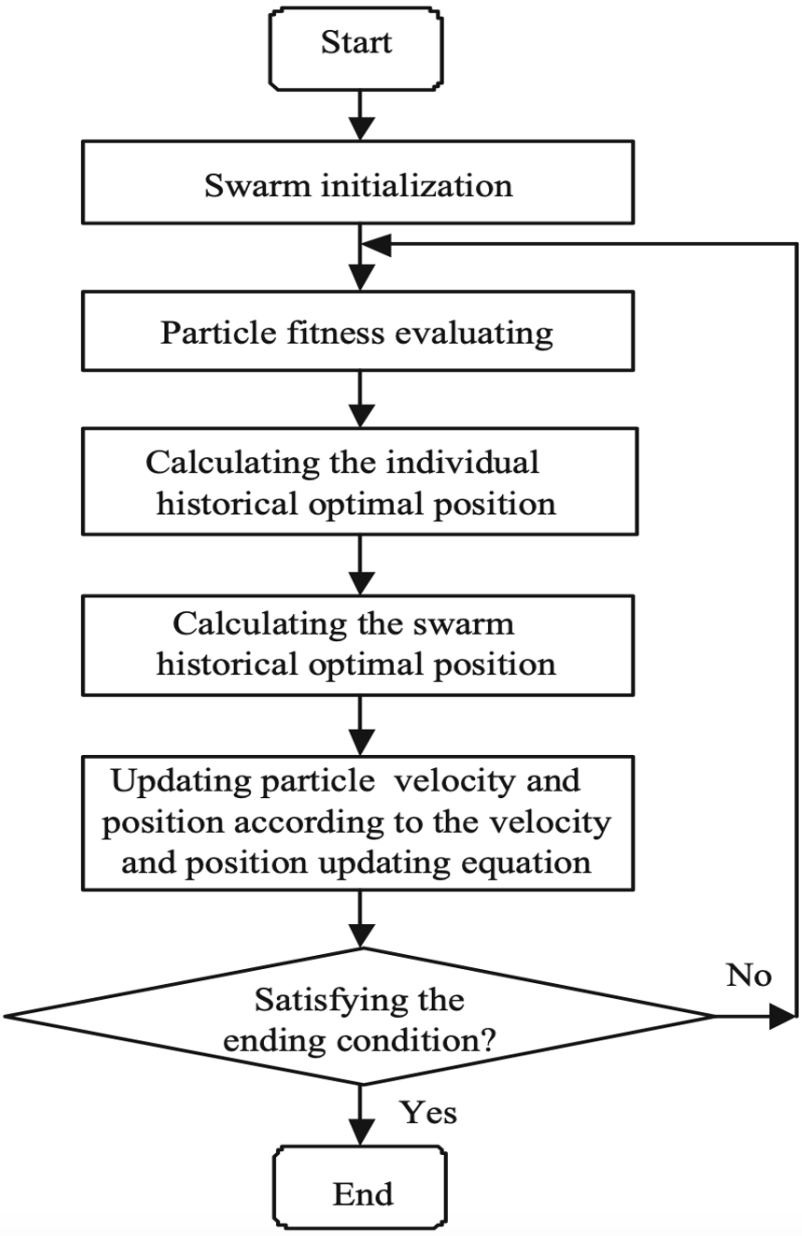


Figure 4.2.1: Particle swarm optimization [58]

$$v_{i,t+1}^d = v_{i,t}^d + c_1 * rand \times (p_{i,t}^d - x_{i,t}^d) + c_2 * rand \times (p_{g,t}^d - x_{i,t}^d) \quad (4.2.3)$$

$$x_{i,t+1}^d = x_{i,t}^d + x_{i,t+1}^d \quad (4.2.4)$$

Canonical PSO algorithm This formula was modified (Shi and Eberhart 1998) by using interia weight applied for velocity above mentioned formula [58]:

$$v_{i,t+1}^d = w * v_{i,t}^d + c_1 * rand \times (p_{i,t}^d - x_{i,t}^d) + c_2 * rand \times (p_{g,t}^d - x_{i,t}^d) \quad (4.2.5)$$

Another version of PSO was introduced in 2002 by Clerc and Kennedy to improve the convergence by adding a new constriction factor χ so the above formula became more completed [58]:

$$v_{i,t+1}^d = \chi(v_{i,t}^d + \phi_1 * rand \times (p_{i,t}^d - x_{i,t}^d) + \phi_2 * rand \times (p_{g,t}^d - x_{i,t}^d)) \quad (4.2.6)$$

The two equations can be identical by selecting the appropriate parameters.
In the global version of PSO, pbest and gbest have the effect, but in the local version, pbest and nbest(all the particles' optimal position) consider and gbest has no effect. In the below, we have a formula for the velocity update position for the local version (pl refers to the local neighborhood optimal position) [58]:

$$v_{i,t+1}^d = w * v_{i,t}^d + c_1 * rand \times (p_{i,t}^d - x_{i,t}^d) + c_2 * rand \times (p_{l,t}^d - x_{i,t}^d) \quad (4.2.7)$$

In the update formula, the first part of the above formula defines the effect of the velocity which belongs to the previous state of a particle that is memorized. It means the particle has confidence in its current moving state and conducts inertial moving accordingly, so it defines w (interia weight).

The second part of the formula shows the current position distance of the particle and its optimal position(the "cognitive" item that is the result of experience). Parameter c1 is the cognitive learning factor or cognitive acceleration factor).
The third part defines the particle's current distance and the global (or local) optimal position in the swarm (social factor). The communication between the particles for cooperation (particles moving) is related to the experience of other particles in a community, so the good particle will move using cognition, so the parameter c_2 (social learning factor or social acceleration factor) is defined. Figure 4.2.2 shows an iterative model example for the PSO.

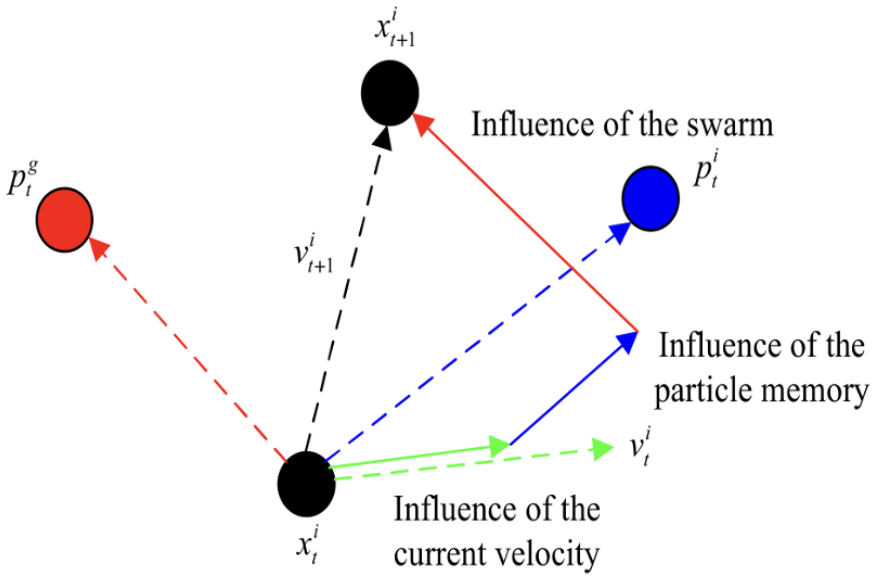


Figure 4.2.2: Iterative Model of PSO
[58]

4.2.2 Ant Colony Optimization Algorithm (ACO)

The Ant Colony Optimization (ACO) algorithm is inspired by the collective foraging behavior of ants [59]. Like other swarm intelligence algorithms, this algorithm is inspired by nature. In this algorithm which was first introduced in the 90s, the social behavior of ants is considered as a solution to find the shortest paths or directions. The foraging behavior of Ant colony optimization (ACO) is a base for solving so many problems [**?**] as ants deposit a kind of substance called a pheromone to guide the other ants and to show the best directions. The best path should be followed by the other ants till they converge on the same line. This way, the marked paths become favorable for the others. We have this equation for probabilistic solutions for selecting the next component (transition probability):

$$P(c_i^j|S_p) = \frac{\tau_{ij}^{\alpha}.[\eta(c_i^j)]^{\beta}}{\sum_{c_i^l \in N(s_p)} \tau_{il}^{\alpha}.[\eta(c_i^l)]^{\beta}}, \tag{4.2.8}$$

for $c_i^j \in N(s_p)$ Partial solution is s_p Solution component c_i^j, proportional to $[\tau_i^j]^{\alpha}.\eta(c_i^j)^{\beta}$, where: $\alpha, \beta > o$
where η is a function that assigns to each valid solution component that is depending on the current step—a heuristic value (The heuristic information) [**?**].

4.2.3 Bee Colony Algorithm

This algorithm was introduced in 2005 inspired by the foraging behavior of the honey bees in a colony. The colony has three types of bees: employed bees, onlookers, and scout bees. The task of the employed bees is to find the closest food source with higher nectar. These kinds of bees are also called forager bees. When they go back to the hive, they dance in this area. This dance which is called waggling dance gives some information like directions, distances, and the quality of the nectars (food source) to the onlooker bees which are also called

observers [60]. If the bees find higher quality foods, they forget the information about the previous one and replace it with the position of the new one that has more quality [60]. So having more quality means having more chances to be selected by onlookers. In this algorithm, position means the possible solutions [60].
The scout bees are the version of employed bees that abandon their food sources and search for new ones. They start a foraging process because they search randomly for promising patches. Flowers with more nectar that can be collected with less effort can be met by more bees, whereas flowers with less nectar receive fewer bees [60]. In this algorithm, the number of food sources is equal to the number of employed bees, and employed bees give the information to the onlookers. There are some equations like the probability Pi of the onlookers to choose the food source. As far as f(x) is a function and x is a location, F(x) is the amount of nectar, and S is the number of food sources.
Initialization Phase When $\vec{X_m}$ is a food source, such $m = 1...SN$ and SN describes the population size, For each x_m vector with n variables, $(x_{mi}, i = 1...n)$, the function is being optimized. The l and u are the lower bound and upper bound of $\vec{X}_{ml}$ [61]. The following definition can be used in this phase (5)[62]:

$$\vec{X}_{ml} = l_i + rand(0,1) * (u_i + l_i) \tag{4.2.9}$$

Employed bees phase can determine a neighbour food source using $\vec{v}_{mi}$ (6) [62]:

$$\vec{v}_{mi} = \vec{x}_{mi} + \phi m_i(x_{mi} - x_{ki}) \tag{4.2.10}$$

Where i is an index (randomly chosen), x_k is a selected food (randomly), ϕ_{mi} is between $[-a, a]$ that is selected randomly. Employed bees share information with onlookers. The $F(x_i)$ of solutions is defined by:

$$Fit_I = \begin{cases} \frac{1}{1+f_i} & if\ Fit_I \geq 0 \\ 1 + abs(f_i) & if\ Fit_I < 0 \end{cases} \tag{4.2.11}$$

Onlooker Bees Phase Onlooker bees are unemployed as scouts. The bees wait for the employed bees and then they choose their food source based on probability. They can use a selection technique such as the roulette wheel selection method (Goldberg, 1989). The probability value p_m (8) [61]:

$$p_i = \frac{F(x_i)}{\sum_{j=1}^{s} F(x_j)} \tag{4.2.12}$$

After a food source $\vec{x}_m$ for an onlooker bee is probabilistically chosen, a neighborhood source is $\vec{v}_m$ determined by using equation (4.3. 2), and its fitness value is computed. As in the employed bees phase, a greedy selection is applied between $\vec{v}_m$ and $\vec{x}_m$ [63].
Scout Bees Phase Scouts are a kind of unemployed bees and they select food randomly. If the solution that is found is abandoned and cannot improve, employed bees become unemployed or scout bees, and they start to search randomly for a new food source.

4.2.4 Firefly Optimization Algorithm

According to the flashing behavior of fireflies, the author Xin-She Yang introduced the firefly algorithm in 2008 [64] using a nonlinear method by combining the light absorption (the exponential decay) and light variance (inverse-square law) considering the distances. The equation that stands for X_i (position) that is a solution is:

$$x_i^{t+1} = x_i^t + \beta_0 e^{-\gamma r_{i,j}^2}(x_j^t - x_i^t) + \alpha\varepsilon_i^t \tag{4.2.13}$$

In general, Metaheuristics have some properties such as they propose some searching strategies to solve problems (simple or complex) by finding the optimal solutions. For this purpose, they explore the space, and sometimes the answers are near-optimal. The results are non-deterministic and approximately.
The definition of fireflies:

fireflies are a kind of insect with light that blinks at night. This light is called bioluminescence used to attract other fireflies or for hunting purposes. Sometimes that is used as a warning about predators. There are some assumptions for this metaheuristic algorithm:
Fireflies are unisexual so all types can be attracted to each other, but the level of this attraction is related to the amount of their brightness so the less bright ones always become attracted to the brightest one, furthermore, there is a relation between attractiveness between closeness(distance) and attractiveness of a firefly so that the closer fireflies are more attractive, it means they are brighter, so they attract the other fireflies and this distance reduces, therefore, it is said that distance between these two fireflies is less than the others and vice versa. When they are far away, they have less attractiveness. The below equation summarizes this assumption:

$$Attractiveness = \frac{1}{Distance}$$

But sometimes the two have the same brightness. In this case, the other firefly will select one of them randomly, no matter which one. The brightness is associated with the function that wants to be solved. They can subdivide into smaller groups such that each subgroup swarms around the model which makes them suitable for optimization and NP-hard problems. The behavior of FA is simple is suitable to solve continuous problems. As far as each firefly can work without dependency, they become good for parallel implementation. The less attractive fireflies move to the more attractive which are the brighter ones.

The firefly attractiveness: The attractiveness is its brightness intensity (I) between the two fireflies when the distance is r_{ij} and $I_s$is the intensity at the source (intensity at source) is defined:

$$I(r) = \frac{I_s}{r^2}$$

Based on the inverse square law, we have: $I(r) = I_0 e^{-\gamma r^2}$

Where we have n fireflies, x_i solutions for the i_{th} firefly, this equation corresponds to the objective function of i_{th} firefly. The brightness I show the last position of the fitness value or f(x):

$$I_i = f(x_i) \tag{4.2.14}$$

β is the attractiveness value based on r_{ij} [65], so β is the attractiveness at r=0. Where the light absorption coefficient is described by γ, the attractiveness function is:

$$\beta(r) = \beta_0 e^{-yr^2} \tag{4.2.15}$$

For the movement of fireflies from the position x_i to x_j we have the below equation [90] when $\alpha\varepsilon$! a random parameter, $\beta_0 e^{-\gamma r_{ij}^2}(x_i - x_j)$ is due to the attraction of the j_{th} firefly and $\beta_0 = 0$ shows a simple random movement:

$$x_i(t+1) = x_i(t) + \beta_0 e^{-yr_{i,j}^2}(x_i - x_j) + \alpha\varepsilon_i \tag{4.2.16}$$

Based on the FA algorithm, the attractiveness of the new position compares with the previous one. If the new position produces a higher β, the firefly will move to the new position and vice versa. The termination criterion of the algorithm can be based on the iterations or predefined fitness value $f(x)$ [66].
The brightest firefly moves randomly as [66]:

$$x_i(t+1) = x_i(t) + \alpha\varepsilon_i \tag{4.2.17}$$

As depicted in the tables, the introduced FA was used to solve the TSP problem. It showed that FA average of best tour for three datasets. As far as the fireflies coming together in a neighborhood to make a couple, we can say that FA is inspired by the graph coloring problem. Figure 4.2.3 summarizes the algorithm in three steps where N is the number of initial solutions, i is the number of iterations, and I indicates the brightness (intensity) [67].

Algorithm 1: Steps of FA

Result: Best Solution Available

Let there are N initial solutions and i denotes the number of iteration, with 'max-itr' maximum iteration number. 'I' is the intensity or brightness of flash;

1. Generate initial solution 'N' randomly.;

2.**for** $i \leftarrow 1$ **to** *max-itr* **do**

    Calculate Brightness I;

    sort I such as $I_j >= I_{j-1}$, for each j;

    for $j \leftarrow 1$ **to** $N - 1$ **do**

        for $k \leftarrow j + 1$ **to** N **do**

            if $I_k > I_j$ **then**

                Firefly 'j' move in the direction of Firefly 'k';

            end

        end

    end

    Move firefly 'N' in random manner;

end

3.Store the best solution

Figure 4.2.3: Steps of Firefly Algorithm [67]

4.2.5 Krill Herd (KH) Optimization Algorithm

The general framework of FA is based on the initial population, fitness function evolution, and stopping criterion that if it meets the requirements the algorithm stops with the global best solution, otherwise, it starts from the fitness function and repeats.
These algorithms are based on the selection of the fittest that have evolved by natural selection over millions of years used for global optimization problems (global minima) [28]. This population-based algorithm is based on the simulation of the herding of the krills in response to specific biological and environmental processes (Gandomi and Alavi 2012). The coefficients are obtained from the nature. The cost function of each krill that should be minimized is defined as its distances from food and the highest density of the swarm [28]. Three essential actions considered to specify the time-dependent position of an individual krill are [28]:
1) movement induced by other krill individuals,
2) foraging activity, and
3) random diffusion.
The KA algorithm describes a swarm of krills in an n-dimensional space where each krill individual (KIs) looks for the closest foods in a denser swarm, so the variable of location matters and distance till the food is considered as cost function (Gandomi and Alavi 2012; Mandal et al. 2014) and the algorithm considers as a multi-objective. This algorithm which uses genetic operands for the precision of the modeling can be categorized into three: (1) Evolutionary algorithm (EA) (2) Swarm intelligence (SI) (3) Bacterial foraging algorithm (Bolaji et al. 2016). Krills can make large swarms and if predators attack krill, they remove individual krill, So their density reduces. The assumption is the closer the distance to the high density and food, the less the objective function.

$$\frac{dX_i}{X_t} = N_i + F_i + D_i \tag{4.2.18}$$

Tables 4.1 and 4.2 show some of the used parameters related to the

Krill Herd algorithm.

1) Motion induced by other krill individuals (KI)
The KI's try to keep their high density and move due to their mutual effects. The direction of motion induced, α_i, is calculated from the local density of the swarm which is called the local effect, a target density which is called the target effect, and a repulsive density (repulsive effect). This movement can be defined as:

$$N_i^{new} = N_\alpha^i + w_n N_i^{old} \tag{4.2.19}$$

where,

$$\alpha_i = \alpha_i^{local} + \alpha_i^{target} \tag{4.2.20}$$

α_i	direction of motion induced
N^{max}	the maximum induced speed
x	the inertia weight of the motion induced in the range [0, 1]
N^{old}	is the last motion-induced
a^{local}	the local effect provided by the neighbors
a^{target}	the target direction effect provided by the best KI

Table 4.1: Parameter Definitions for the krill herd algorithm [28]

Kbest	Best fitness value by now
Kworst	worst fitness values by now
Ki	the fitness or the objective function value of the ith KI
Kj	fitness of jth (j = 1, 2, . . ., NN) neighbor
X	the related positions
NN	number of the neighbors
ds, i	sensing distance for the ith krill individual
N	number of the KI S
ϵ	a small positive number to avoid singularity
C_{best}	The coefficient of the best KI with the most effect
rand	Random value between [0,1] for boosting exploration
I	The actual iteration number
I^{max}	The max of iterations
V_f	The speed in foraging movement
W_r	Interia weight for the foraging motion [0,1] in the last motion
β_i^{food}	The attraction of the food
β_i^{best}	The best fittest value of a KI
C^{food}	food coefficient
K_i^{best}	The best location of the previous iteration for the ith KI
τ	Randomly chosen directional vector [-1, 1]
Dmax	the maximum diffusion speed between [0.002, 0.010] (ms^{-1})
C_t	constant [0,2]
NV	total number of variables
LB_j	lower bound of the jth variables ($j = 1, 2, ..., NV$)
UB_j	upper bounds of the jth variables (j = 1, 2, . . ., NV)
Cr	Crossover probability

Table 4.2: Some other Parameters for the Krill herd algorithm [28]

According to the measured values of the maximum induced speed, is 0.01 (ms^{-1}) [68]. The effect of the neighbors in a krill (attractive/repulsive tendency) movement individual is determined as follows formula:

$$\alpha_i^{local} = \sum j = 1^{NN} \hat{K}^{i,j} \hat{X}_{i,j} \tag{4.2.21}$$

$$\hat{X}_{i,j} = \frac{X_j - X_i}{\|X_j - X_i\| + \epsilon} \tag{4.2.22}$$

$$\hat{K}_{i,j} = \frac{K_i - K_j}{K^{worst} - K^{best}} \tag{4.2.23}$$

There are some unit vectors and some normalized fitness values (can be negative or positive). The vectors show the induced directions by different neighbors. Each value presents the effect of a neighbor(be attraction or repulsion).
For neighbor selection, a ratio can be assigned to find the number of the nearest KIs. Using the actual KI s, a sensing distance (ds) should be determined around a KI as it is in the figure below to find the neighbors. if the distance of the two KI s is less than the specified $d_{s,i}$, they are neighbors.

$$d_{s,i} = \frac{1}{5N} \sum_{j=1}^{N} \|X_i - X_j\| \tag{4.2.24}$$

'The known target vector of each krill individual is the lowest fitness of an individual krill, so we have the below equation for the global optima using α^{target} [28]:

$$\alpha^{target} = C^{best} + \hat{K}_{i,best} \hat{X}_{i,best} \tag{4.2.25}$$

Here is the formula of Cbest for the above equation:

$$C^{best} = 2(rand + \frac{I}{I_{max}}) \tag{4.2.26}$$

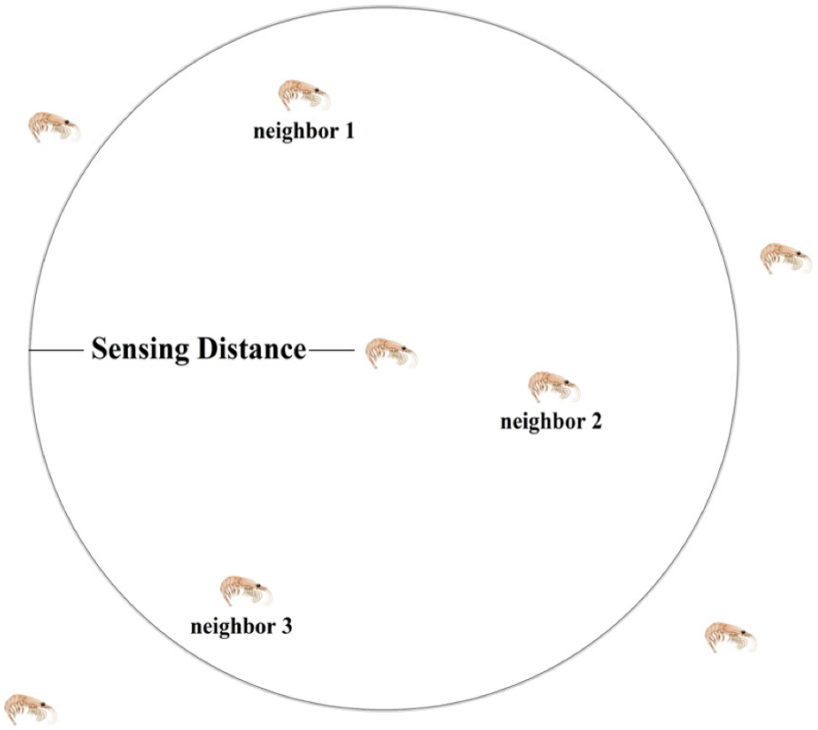


Figure 4.2.4: Flowchart of the Krill herds algorithm
[28]

2) Foraging motion

The foraging motion for the ith KI is formulated where we have two parameters which are food location, and previous experience:

$$F_i = v_f \beta i + w_f F_i^{old} \tag{4.2.27}$$

where,

$$\beta i = \beta_i^{food} + \beta_i^{best} \tag{4.2.28}$$

The food effect is defined in terms of its location. The center of food should be found first and then try to formulate food attraction. This

cannot be determined but can be estimated. In this study, the virtual center of food concentration is estimated according to the fitness distribution of the krill individuals, which is inspired by the "center of mass". The virtual center of the food is based on their fitness (center of mass theory) estimated for finding the location at first, so the food attraction should be formulated later on. Krills are depicted in figure 4.2.4. We have the below for each of the iterations [28]:

$$X^{food} = \frac{\sum_{x=1}^{N} \frac{1}{K_i} X_i}{\sum_{x=1}^{N} \frac{1}{K_i}} \tag{4.2.29}$$

The food attraction ith KI is as follows:

$$\beta_i^{food} = C^{food} \hat{K}_{i,food} \hat{X}_{i,food} \tag{4.2.30}$$

Because the effect of food in the krill herding decreases during the time, the food coefficient is determined as:

$$C^{food} = 2(1 - \frac{I}{I_{max}}) \tag{4.2.31}$$

The food attraction is defined to possibly attract the krill swarm to the global optima. Based on this definition, the krill individuals normally herd around the global optima after some iteration. This can be considered as an efficient global optimization strategy that helps improve the globality of the KH algorithm. The effect of the best fitness of the ith KI can be calculated as:

$$\beta_i^{best} = \hat{K}_{i,best} \hat{X}_{i,ibest} \tag{4.2.32}$$

3) Physical diffusion
This process is random in terms of speed for maximum diffusion and a random directional vector. The formulation is:

$$D_i = D_{max} \delta \tag{4.2.33}$$

The better the position of the krill is, the less random the motion is [69]. The below formula shows the decreases of the random speed with time, linearly:

$$D_i = D_{max}(1 - \frac{I}{I_{max}})\delta \tag{4.2.34}$$

It is proved that in these three motions, if the fitness value corresponds to each of these factors: $(K; Kbest; Kfood)$ is better (less) than the fitness of the i_{th} krill, it has an attractive effect; and vice versa. The position vector of a KI for $[t, t + \Delta t]$ interval is defined by the following equation:

$$X_i(t + \Delta t) = X_i(t) + \Delta t \frac{dX_i}{dt} \tag{4.2.35}$$

Δ t is an important constant used for scaling of the speed vector depending on the search. The absolute of the subtraction of upper bound and lower bounds shows the search space. The search becomes more precise for the KI s if Ct takes the lower value.

$$\Delta t = c_t \sum_{j=1}^{NV} (UB_j - LB_j) \tag{4.2.36}$$

Genetic operators for the Krill algorithm The genetic algorithm is embedded in the KH algorithm, and the adaptive genetic reproduction approaches use crossover and mutation which are inspired from the classical differential evolutionary algorithms.
1. Crossover is used for global optimization, and here an adaptive vectorized crossover is applied to improve the performance. The crossover is controlled by a probability called Cr.
Actual crossover can be performed in binomial or exponential forms. For the binomial form, the crossover operand applies for each of the components for a uniform distribution random number [0, 1], then the m_{th} value of Xi, $x_{i,m}$, where $r \in 1, 2, ..., i-1, i+1, ..., N$ drives

from the below:

$$x_{i,m} = \begin{cases} x_{r,m} & rand_{i,m} < Cr \\ x_{i,m} & else \end{cases} \tag{4.2.37}$$

$$Cr = 0.2\hat{K}_{i,best} \tag{4.2.38}$$

2. Mutation: There is another probability that applies for mutation in the KH algorithm, called MU, as in the below: where: $p, q \in 1, 2, ..., i-1, i+1, ...K$, μ is a number [0, 1] when in $\hat{K}_(i, best)$ the nominator is $k_i - k^{best}$ [70]:

$$x_{i,m} = \begin{cases} x_{gbest,m} + \mu(x_{p,m} - x_{q,m}) & rand_{i,m} < MU \\ x_{i,m} & else \end{cases} \tag{4.2.39}$$

$$Mu = 0.05\hat{K}_{i,best} \tag{4.2.40}$$

The steps of the Krill Herd optimization algorithm are:

1) Data Structures: Define the simple bounds, determination of algorithm parameter(s), etc.

2) Initialization: Randomly create the initial population in the search space.

3) Fitness evaluation: Evaluation of each krill individual according to its position.

4) Motion calculation:

o Motion induced by the presence of other individuals,

o Foraging motion

o Physical diffusion

5) Implement the genetic operators

6) Updating: updating the krill individual position in the search space.

7) Repeating: go to step III until the stop criteria are reached.

8) End

On the next page, Figure 4.2.5 shows the flowchart of the KH algorithm.

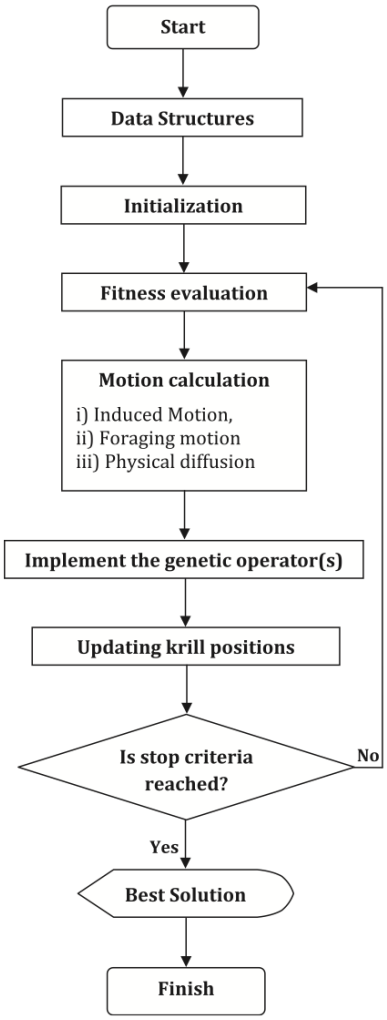


Figure 4.2.5: Krill Herd optimization algorithm [28]

4.2.6 Flora Optimization Algorithm

This algorithm which was mainly used in my first research and I presented at a conference, has some inspirations from the nature of other swarm intelligence algorithms. The main concentration of this algorithm is based on the movement of the seeds and their offspring which can lead to the evolution, distinction, or creation of a new plant. As the offspring move to a new place, based on the characteristics of that environment (fitness) like climate, the seeds behave differently:
1) Sometimes, they can't adapt to that environment so they can't survive. They are distinct.
2) Some of them become adapted to the new environment and during the time they evolve, they become original plants by themselves for the new generations, and the cycle repeats, like spreading offspring.
3) Some of them become a new plant (called rebirth).

The four important elements of this algorithm are [71]:
1) Original Plant- They can spread the seeds in any place in the propagation Distance. It helps for the local search capacity of the Flora algorithm
2) Offspring Plant (Seeds of the original plants)
3) Plant Location
4) Propagation Distance or spreading distance means learning from the previous original plants and refers to how far a seed can go. In spreading behavior, always the past generation movements are considered to update the solution that helps the algorithm avoid running into the local extremum.
The spreading is constant, so flora can migrate until they find the best answer (fittest area).
It can take the local optimal position as the center to explore around space, so It can converge to optimal fast. Spreading the seeds is done randomly but in a special radius. The seed dispersal of the plants has two modes:
1) Autochory (plants that spread by themselves like mechanical prop-

agation)
2) Allochory: in this way, the plants spread through external forces like biological propagation, anemochory, and hydrochory.
Allochory provides the conditions for plants to migrate to farther uncharted regions such that the direction and distance are determined as the wind changes, so by increasing the scope of exploration of flora, the possibility of extinction reduces [71].
The survival probability:
1) For a suitable environment: a plant survives and after being ripe spreads the seeds.
2) For harsh environments we have:
a) The probability of adapting to the new environment where a plant evolves.
b) The extinction of that flora in that region.
But before the distinction of flora [71], there is a probability of multiplication of flora in other areas, because the seeds may be moved to any new environment where the flora regains reproduction. As the propagation type is multi−generation, the flora has the chance of finding the best (optimal) growth environment.
The three main patterns in flora:
1) evolution behavior, which means the evolving probability to adopt
2) spreading behavior, which means movements of seeds: Autochory, Allochory
3) select behavior, Which means survival or distinction of the flora.
There are some specifications in the form of rules to explain the above-mentioned behaviors:
Rule 1: A species can be the primitive one in a new environment after a random distribution of the seeds. In Figure (4.2.5), original plants were distributed randomly in the area, as the $\diamond$ (x1).
Rule 2: If a plant adapts to the environment, it should evolve, and during this evolution, its hierarchy is based on the propagation distance of the last two nearest generations, so this hierarchy is not complete.
Rule 3: The range of the seeds distribution area is considered a

circle. The radius r of this circle is equal to the maximum propagation distance. Every place in this circle with a radius of r and its circumference is for the distribution of the seed. In the figure (4.2.5), the seeds are distributed in the propagation distance. Distance 1, Distance 2, and Distance 3 define the propagation distances, and the offspring is described as (a1,a2,a3).

Rule 4: The probabilities of survival for the plants are different since the environmental factors for different areas are not the same. This probability is related to the plant and how well it is fitted to the environment, so more fitted plants have a higher survival probability, but in some cases, some inter-specific competition can change this rule. The solid line in the figure depicts a living plant and the dotted line depicts that the plant is died [71].

Rule 5: When a seed spreads to a very far environment, the probability of survival reduces, because the new environment has some new properties like climate change that are possibly much more different than the environment of the original plant. As we have:

$$d \propto \frac{1}{p}$$

Based on this rule: $fitness(a1) > fitness(a2) > fitness(a3)$, because of their distance from the original plants, the fittest one is the closest one to the original plants. Because of competition in the figure (as an exceptional case), offspring a1 died even though it had a high fitness value, but a2 became an original plant that can spread its seeds.

Rule 6: There is a boundary for seeds spreading, in a way that the distance of the seeds should not become more than the maximum limit. These limit areas define constraints.

In Figure (4.2.6), $\square(b1, b2, b3)$ are described as the new plants, and two of them survived as b1 and b3, but b2 did not. The selection between the two surviving plants b1 and b3 is random. In the figure (4.2.6), b1 has been selected as the latest one. Distance 2 is for a2, and distance 3 is for b1. Distance 2 is based on distance 1, and distance

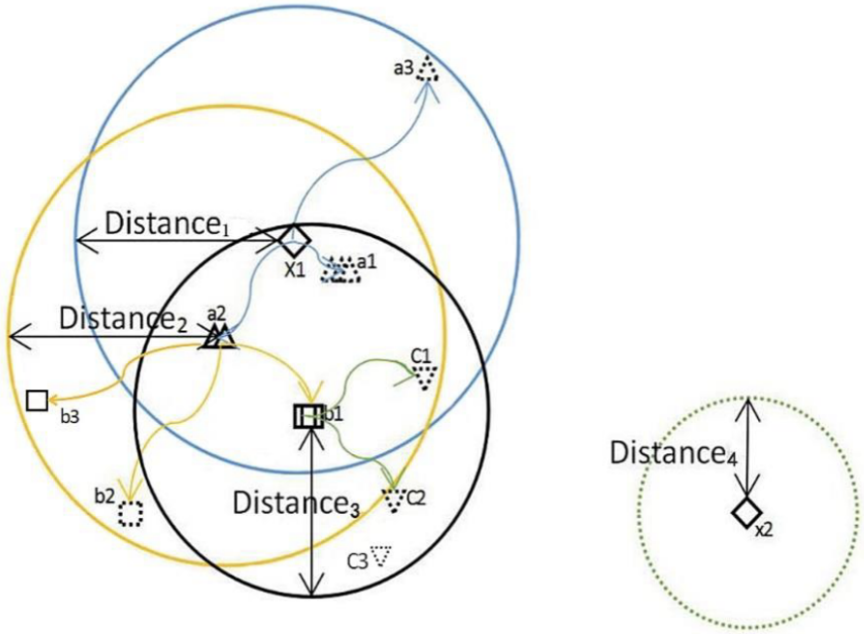


Figure 4.2.6: The Process of Migration and Reproduction [71]

3 is learning from the previous distances 1, and 2. If b1 which is selected randomly between b1 and b3, spread, the distances will be based on distance2, 3. If all the offspring plants die, as $(c1, c2, c3)$, a new original plant can be generated randomly.

Evolution Behavior

Based on the theory, the propagation distance (eq. 4.2.41) is evolved from the parent and grandparent plants. Table 4.3 defines some parameters used in the formulas.

$$d_j = d_{1j} \times rand(0,1) \times C_1 + d_{2j} \times rand(0,1) \times C_2 \qquad (4.2.41)$$

d_j	propagation distance
d_{1j}	propagation distance of grandparent plant
d_{2j}	propagation distance of new parent plant
c1 and c2	Learning Coefficents
rand(0,1)	independent uniformly distributed number
$d_{1j}^{'}$	new grandparent propagation distance
d_{r}^{2j}	new parent propagation distance
P	Position
N	Number of solutions and Plants
d	maximum limit area

Table 4.3: Parameters [71]

The new grandparent propagation distance is defined as:

$$d_{1j}^{'} = d_{2j} \tag{4.2.42}$$

The new parent propagation distance is the standard deviation between the positions of the original plant and the offspring plant:

$$d_{2j}^{'} = \sqrt{\frac{\sum_{i=1}^{N}(P_{i,j} - P_{i,j}^{'})^2}{N}} \tag{4.2.43}$$

Spreading Behavior The first generation of the plants is randomly selected such that N plants for N solution. P defines the position of the original plant in the form of a matrix $P_{i,j}$ (where i refers to the dimension, j is the number of plants in the flora, and d is the maximum limit area).

$$P_{i,j} = rand(0,1) \times d \times 2 - d \tag{4.2.44}$$

The position of the offspring plant [71]:

$$P_{i,j\times m}^{'} = D_{i,j\times m} + P_{i,j} \tag{4.2.45}$$

where m is the number of seeds that one plant can propagate, we have Table 4.4 that defines the spreading behavior parameters.

$P'_{i,j\times m}$	The position of offspring plant
$P_{i,j}$	The position of the original plant

Table 4.4: Parameters [71]

If no offspring plant survives, then a new original plant is generated according to the:

$$P_{i,j} = rand(0,1) \times d \times 2 - d \tag{4.2.46}$$

Select Behavior
The survival probability is summarized as in below:

$$P = \left| \sqrt{\frac{F(P'_{i,j\times m})}{F_{max}}} \right| \times Q_x^{(j\times m-1)} \tag{4.2.47}$$

Where our objective function is the fitness equation, we have Table 4.5 which defines the other parameters for the select behavior.
Deciding whether a plant survives or not is based on the proportion

Q_x	the selective probability
F_{max}	Max Fitness of Flora
$F(P'_{i,j\times m})$	Fitness of the jth answer

Table 4.5: Some other parameters [71]

select method or roulette wheel method. Selection is based on score values and accepting probability. A higher score means greater probability.
r which is a random number in [0,1] becomes generated (uniform distribution) every time, and the offspring plant will be alive if the survival probability P is bigger than r ($P > r$), or it will die. N offspring plants become selected among the alive offspring as new

R	A random number between 0 and 1
P	Probability between 0 and 1

Table 4.6: parameters
[71]

original plants and the above behaviors repeat until the accuracy requirement is reached or the maximum number of iterations is achieved [71].

The main steps of artificial flora are as follows [71]:
(1) Generation of N original plants based on $P_{i,j} = rand(0,1) \times d \times 2 - d$;
(2) Calculate propagation distance according to equations:

$$d_j = d_{1j} \times rand(0,1) \times C_1 + d_{2j} \times rand(0,1) \times C_2;$$

;

$$d'_{1j} = d_{2j};$$

$$d'_{2j} = \sqrt{\frac{\sum_{i=1}^{N}(P_{i,j} - P_{i,j})^2}{N}};$$

(3) Generation of offspring plants by:

$$P'_{i,j\times m} = D_{i,j\times m} + P_{i,j};$$

and calculate the fitnesses; (4) Calculate the survival probability of offspring and roulette wheel for selection:

$$P = \left| \sqrt{\frac{F(P'_{i,j\times m})}{F_{max}}} \right| \times Q_x^{(j\times m-1)};$$

(5) For the surviving plants, randomly selection of N new original plants. If they don't survive, generate new original plants by:

$$P_{i,j} = rand(0,1) \times d \times 2 - d;$$

(6) If the new original plant's solutions are better than the previous one, then succeed them and save the best answer;
(7) Estimate whether the results are meeting the termination criteria or not. It can be decided by the accuracy requirement or the number of iterations. If so, then it is the optimal solution, otherwise, the algorithm starts from step number 2;
Where M and N are: M: Maximum branching number (the number of seeds that one original plant can produce), and N: Number of the original plants, the time complexity of this algorithm becomes O(NM). This algorithm has been tested by benchmark functions like Sphere, Rosenbrock, Rastrigin, Schwefel, Griewank, and Ackley to check its accuracy, and is compared for Particle Swarm Optimization (PSO) and Artificial Bee Colony (ABC) algorithms and different datasets. The results concluded that the AFO can have higher accuracy and stability. The reasons are:
1)Every time, It takes the surviving offspring plants as the new original plants, so the local optimal location becomes the center and it can explore the space around and converge faster.
2) As far as the direction of spreading the seeds and distances are within a circle (the seeds can be taken to any part of this propagation environment), there is always a local search guarantee.
3) The algorithm can ignore the local optimum and focus on global searching, because based on the algorithm, when there is no better offspring, it randomly generates the new original plants.

4.2.7 Whale Optimization Algorithm

This section explains a meta-heuristic optimization algorithm (Whale Optimization Algorithm, WOA) found by Mirjalili in 2016 based on the hunting behavior of whales. Humpback whales which are amongst [72] one of the biggest whales hunt prey like krill and small fish herds that are close to the surface in a spiral manner. Their method which is called the bubble-net feeding method is based on the foraging behavior of whales that lead to the creation of bubbles along

a circle like **'9'** shape. These events were captured by tag sensors for 9 whales. The two maneuvers of whales discovered by scientists are called 'upward-spirals' and 'double-loops'. Humpback whales swim down 12 m [72] and start to make bubbles in a spiral shape around the prey and move upward to the surface. The second maneuver consists of three different models: coral loop, lobtail, and capture loop. This bubble-net feeding makes humpback whales unique from other whales. The spiral bubble-net feeding method is mathematically formulated to perform optimization.

Mathematical models for the Whale Optimization Algorithm

The models which are called: encircling prey, spiral bubble-net feeding maneuver, and search for prey are explained for a better understanding of the algorithm.

1) Encircling prey The whale can find the position of the prey, but this position is not always the best position (solution) that is optimal, but in the first step the whale optimization algorithm supposes the first position as the best answer, and then it starts looking for better answers in the next iterations. The next position is called $\overrightarrow{X}(t+1)$ which can be the best answer. At each step, every solution is called a candidate solution.

The updated positions are represented by the following formulas when [73] $\overrightarrow{D}$ represents the distance measurement, t is the recent iteration, the vectors of $\overrightarrow{A}$, and $\overrightarrow{C}$ are coefficients defined in the formula such $\overrightarrow{A}[-a, a]$, $\overrightarrow{X}^*$ is the best candidate solution at each step, $\overrightarrow{a}$ is a decreasing vector [2,0], and $\overrightarrow{r}$ in [o,1] is a random vector.

$$\overrightarrow{X}(t+1) = \overrightarrow{X}^*(t) - \overrightarrow{X}(t) \tag{4.2.48}$$

$$\overrightarrow{D'} = \left|\overrightarrow{C}\overrightarrow{X}^*(t) - \overrightarrow{X}(t)\right| \tag{4.2.49}$$

$$\overrightarrow{A} = 2\overrightarrow{a}.\overrightarrow{r} - \overrightarrow{a} \tag{4.2.50}$$

$$\overrightarrow{C} = 2.\overrightarrow{r} \tag{4.2.51}$$

In 2-dimensional data, (X, Y) is the position of an agent, and $(X*, Y*)$ is the position of the best search agent. By adjusting the values of A and C, the current agents change their position and become updated. The vector r also helps to change to another position in the space in the neighborhood. So, as far as the equation $\overrightarrow{X}(t+1)$ has all these vectors inside, it can be used to update the positions in the encircling prey step. The same concept can be extended to search space for $n - dimensional$ data, the search agents will move in hyper-plane around the best solution achieved so far.

2) Bubble-net attacking method (exploitation phase) This method of hunting that is also called the exploitation phase can be defined with 2 approaches:

1) Shrinking encircling mechanism:

This behavior is done by changing (decreasing) the value of $\overrightarrow{a}$ in the formula of $\overrightarrow{A}$. By selecting a random number for $\overrightarrow{A}$ in the interval $[-1, 1]$, the new position of a search agent can be defined anywhere in between the original position and the current best position of an agent.

2) Spiral updating position:

After the calculation of the distance between the two positions (between the whale and the prey), the spiral equation was created to mimic the helix-shaped movement of whales as:

$$\overrightarrow{X}(t+1) = \overrightarrow{D'}.e^{bl}.cos(2\pi l) + \overrightarrow{X^*}(t) \tag{4.2.52}$$

is the i_{th} distance the (best solution)

where

$$\overrightarrow{D'} = \left|\overrightarrow{X}^*(t) - \overrightarrow{X}(t)\right| \tag{4.2.53}$$

indicates the distance of the i_{th}, b is a constant for logarithmic shape, l $[-1, 1]$ randomly chosen. As far as whales swim in two approaches

simultaneously, we suppose that there is a probability p [0, 1] of %50 to select either the shrinking encircling mechanism or the spiral approach for $\overrightarrow{X}(t+1)$.
if $p < 0.5$, we use:

$$\overrightarrow{X}(t+1) = \overrightarrow{X}^{*}(t) - \overrightarrow{A}.\overrightarrow{D} \tag{4.2.54}$$

otherwise ($P \geq 0.5$), we use:

$$\overrightarrow{X}(t+1) = \overrightarrow{D'}.e^{bl}.cos(2\pi l) + \overrightarrow{X^{*}}(t)$$

3) Search for prey (exploration phase) Exploration alludes to a metaheuristic algorithm's capacity to search in diverse environment ranges to find the ideal solutions [75],[76], [77]. In this method, the whales search randomly based on the position of each other, so we can use $-1 \leq A < 1$ randomly to make them move far away from a reference whale. So we use again $\overrightarrow{A}$ as a [74] vector like in the previous step, but in this case, we use randomly chosen search agents [78], because this is the exploration phase and we search for the prey instead of the best agent so far in the exploitation step to perform a global search. This approach which is based on a random search is modeled below [79]:

$$\overrightarrow{D} = \left|\overrightarrow{C}.\overrightarrow{X}_{rand} - X_i^t\right| \tag{4.2.55}$$

$$\overrightarrow{X}(t+1) = \overrightarrow{X}_{rand} - \overrightarrow{A}.\overrightarrow{D} \tag{4.2.56}$$

The steps of the Whale Optimization Algorithm are [72]:
Step 1: Initialize the whale's population Xi (i = 1, 2, ..., n)
Step 2: Calculate the fitness of each search agent

X_{*}=the best search agent
Step 3: while (t < maximum number of iterations)
for each search agent Update a, A, C, l, and p **Step 4:**if1 (p<0.5)

if2 ($|A| < 1$)

Update the position of the current search agent by $\overrightarrow{X}(t+1) = \overrightarrow{X}^{*}(t) - \overrightarrow{A}.\overrightarrow{D}$

Step 5:else if2 ($|A| \geq 1$)

Select a random search agent
Update the position of the current search agent by: $\overrightarrow{X}(t+1) = \overrightarrow{X}_{rand} - \overrightarrow{A}\overrightarrow{D}$

Step 6: elseif1 ($p \geq 0.5$)
Update the position of the current search by: $\overrightarrow{D'}.e^{bl}.cos(2\pi l) + \overrightarrow{X^{*}}(t)$
end if1
end for

Step 7: Check if any search agent goes beyond the search space and amend it

Step 8: Calculate the fitness of each search agent

Step 9: Update X* if there is a better solution
t=t+1
end while
return X*

$f^{i,j,k,l}(m1, m2, m3)$:

$f^{i,j+1,k,l}(m1, m2, m3) = f^{i,j,k,l}(m1, m2, m3) + C_i \frac{del(i)}{\sqrt{(del^T(i)del(i)}} + ...$

Chapter 5

The Summarization of the Related Papers and the New Methods

5.1 Results of The conference paper

In the conference paper [60] AFO was combined with K-means in an iterative approach and a new clustering method based on the Flora Algorithm presented. The method employed the flora seeds(offspring)to search for the set of cluster centers that minimize the distance. Our dataset was an iris dataset with three outputs (class labels). The three classes are called Setosa, Versicolor, and Virginica, respectively from the iris dataset which is a standard dataset. We supposed that we don't know the output and the algorithms want to find them and check the correctness. So, in the dataset, We eliminated the class labels from the output to perform an unsupervised approach to the dataset. We tested this with other algorithms(ABC and ACO) as well and the accuracy measurement was the main factor based on the classification formula.
The Steps of the Artificial Flora Algorithm with k-means, where the roulette wheel should be applied to decide whether a plant will be

alive or not must are:
1- Initialization by setting the parameters
2- Start with N plant and calculate the population distance using these formulas: propagation distance is:
$d_j = d_{1j} \times rand(0,1) \times C_1 + d_{2j} \times rand(0,1) \times C_2$;

The propagation distance of the grandparent plant is:
$d'_{1j} = d_{2j}$;

The propagation distance of the new parent plant is:
$d_{2j}\prime = \sqrt{\frac{\sum_{i=1}^{N}(P_{i,j} - P'_{i,j})^2}{N}}$

3- Calculate the Euclidean distance between the plant and centroids to check whether they belong to that cluster or not,
Repeat it;
4- Selection of some neighborhood search space in a circular shape, where r is the max distance
5- Update the positions of the offspring (place more offspring in the best areas)
6- Compute the fitness and select the fittest offspring;
7- Keep the assignments of offsprings until they meet the maximum number of iterations.

The tables show the confusion matrix for the three classes and how they are placed in the true or false classes.

classes	1	2	3	total
1	50(33.3)	0	0	100
2	0.0	49(32.7)	0	100
3	0.0	1(0.7)	50(33.3)	98
Percent	100	98	100	99.3(0.7)

Table 5.1: Confusion Matrix of Artificial Flora

We obtained the below results for the artificial bee colony algorithm

N	20
M	10
Iterations	50
p	0.9
c_1	0.75
c_2	1.25
μ	0.2

Table 5.2: Parameters values for Artificial Flora

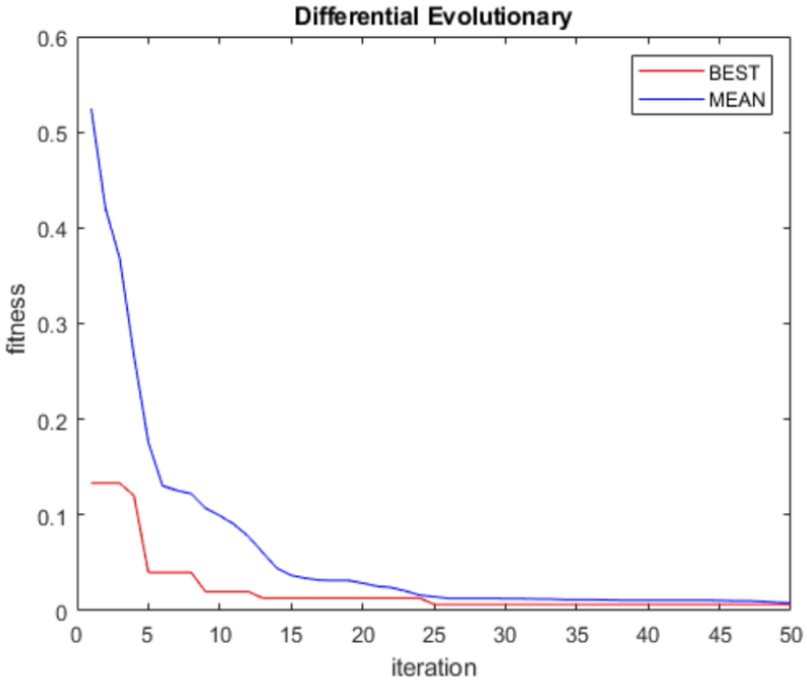


Figure 5.1.1: Best Fitness of AFO in 50 Iterations

classes	1	2	3	total
1	50(33.3)	0.0	0.0	100
2	0.0	48(32.0)	0.0	100
3	0.0	2(1.3)	50(33.3)	96.2(3.8)
Percent	100	96(4.0)	100	98.7(1.3)

Table 5.3: Confusion Matrix of Artificial Bee Colony Algorithm

classes	1	2	3	total
1	50(33.3)	0.0	0.0	100
2	0.0	49(32.7)	0.0	100
3	0.0	1(0.7)	50(33.3)	98(2.0)
Percent	100	98	100	99.3(0.7)

Table 5.4: Confusion Matrix of Ant Colony Algorithm with Alpha=1

(ABC) with NB=20 which is the number of bees (population size):

The conclusion which is the statistics of the accuracy measurement for three algorithms is summarized in the form of Table 5.5.
The average is taken 30 times (each time 50 iterations) for the iris dataset.
The higher accuracy shows the algorithm which had the best performance. We used the supervised techniques for accuracy measurement

Statistics	Min	Avg	Max	Sum	Std.dev
AF	0.88871	0.969359	0.9912	29.08078	0.023425
ABC	0.92436	0.969814	0.98258	29.09443	0.014943
ACO	0.97413	0.984028	0.9912	29.52084	0.004549

Table 5.5: Statistical Measurements for Accuracy
[60]

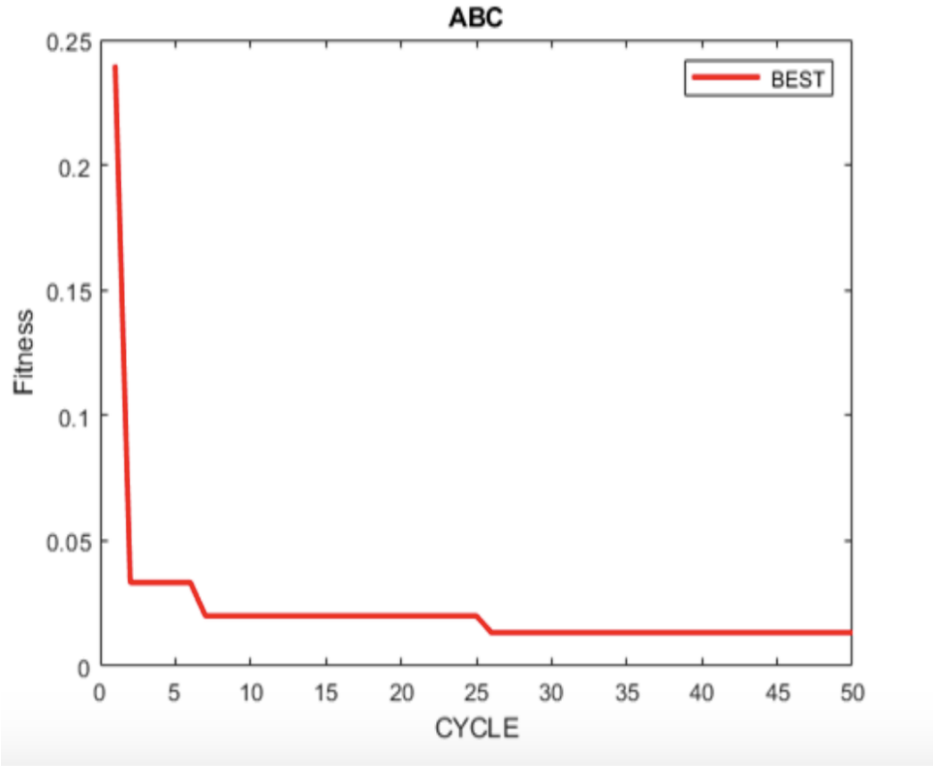


Figure 5.1.2: Best Fitness of ABC in 50 Iterations

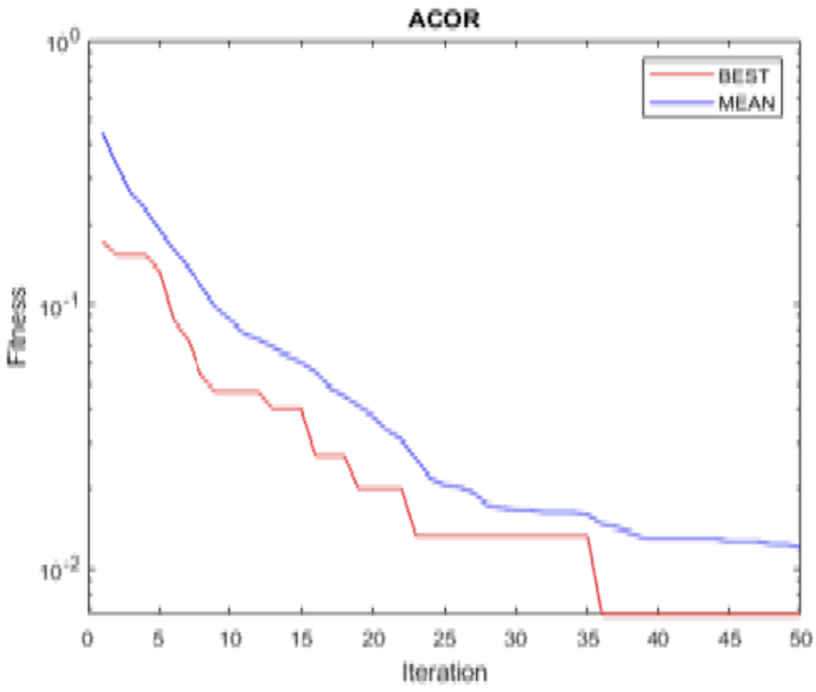


Figure 5.1.3: Best Fitness of ACO in 5O Iterations with Alpha=1

that is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1.1}$$

‘

When TP, TN, FP, and FN represent the True- Positive, True-Negative, False- Positive, and False -Negative, respectively.

5.2 The clustering Method for whale algorithm by k-means

In this article, we applied a clustered and non-clustered approach to solve TSP. The WOA and K-means algorithms combined to improve fitness function. The population size and iteration numbers are 100, and 20 respectively. The number of the cities is 51, 318, and 1323. First, we apply K-means to divide our data into K clusters, then we apply the WOA algorithm to find a minimum tour in each cluster, and then we connect our clusters to find the best solution that is the optimal path. The achievements of this method are proved in the tables.
The procedure is:
i) We calculate the distances between the centroids of each cluster to find the clusters with minimum distances (Euclidean distance is applied for finding the distance).
ii) Combine the selected clusters into one bigger cluster. Then we connect the tours of these two newfound clusters.
iii) We repeat step 1 until the generation of the minimum tour

Algorithm: K-means,
Inputs:
K: cluster numbers or the initial centroids,
D: a set of n objects,
Output: K clusters,

Method:
(1) arbitrarily choose K objects from D,
(2) repeat,
(3) according to the average of the objects within the cluster, (re)assign the objects to the closest one
(4) update by calculating the average of the objects for each cluster, and new assignments
(5) until no change happens in the clusters [31].
The steps of our algorithm are as follows [31]:
Step 1: Initialize the number of the population as shown in Table 1
Step 2: Specify K based on the $k = \sqrt{\frac{N}{2}}$
Step 3: Applying the Kmeans algorithm
Step 4: Applying whale optimization algorithm for i=1:K
Step 5: Find the position of the cities
Step 6: Sorting by indexing
Step 7: Find the nodes (cities) in each cluster that are closer to the centroids of that cluster
Step 8: Join the closest cities to another cluster
Step 9: Stopping criterion till no cluster remains un-joined.
Tables (5.6), and Table (5.7) present the fitness and the timing of the best tour for the clustered and nonclustered methods. The timing and fitness are improved clustered method.
The results of the nonclustered whale algorithm are article: 1176.421, 393928.2, and 8262213 in table 5.6 have improved to 489.8785 [31], 82908.2, and 1.19E+06 in table 5.7 for the clustered approach. The average of the best fitness was considered as benchmark.

The figures 5.2.1 to 5.2.3 present the illustration of the clustered and non-clustered methods to show how the cities are connected. The figures on the right side are more organized and show the clutering method to solve a TSP or a Vehicle routing problem (VRP).

WOA – TSP Fitness	Eil51	Linhp318	Rl1323
Avg	1176.421	393928.2	8262213
Max	1262.614	409489.4	8740947
Min	1100,269	376614.9	8174234.322
Stdev.s	48.53605	8468.606	49003.98
WOA – TSP Timing	Eil51	Linhp318	Rl1323
Avg	3.765785	16.63365	63.50511
Max	4.7562	20.79	79.21
Min	3.1421	13.642	51.669
Stdev.s	0.396958	1.77977	6.286887

Table 5.6: non-clustered Whale Optimization Algorithm for Solving TSP Problem

K-means Method Fitness	Eil51	Linhp318	Rl1323
Avg	489.8785	82908.2	1.19E+06
Max	526.76	88040	1.23E+06
Min	454,9	77273	1.14E+06
Stdev.s	21.2760405	3180.155	2.78E+04
K-means Method Timing	Eil51	Linhp318	Rl1323
Avg	1.2205855	5.707375	24.81385
Max	1.4997	7.953	55.258
Min	1.0021	4.6754	20.345
Stdev.s	0.1708681	0.979334	7.647444

Table 5.7: Clustered Whale Optimization Algorithm for Solving TSP Problem

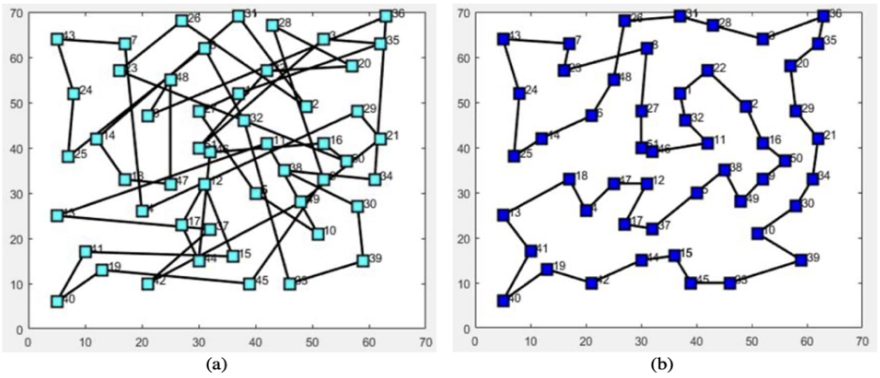


Figure 5.2.1: Figure a) Non-clustered WOA for Eil51 Dataset ; Figure b) Clustered WOA for Eil 51 Dataset

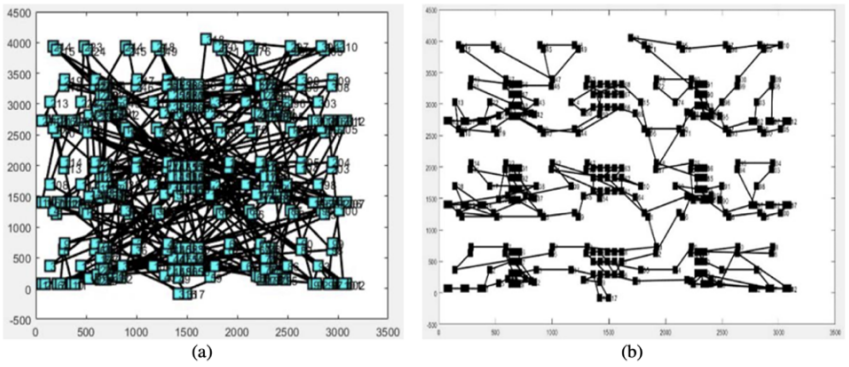


Figure 5.2.2: Figure a) Non-clustered WOA for linhp318 Dataset ; Figure b) Clustered WOA for linhp318 Dataset

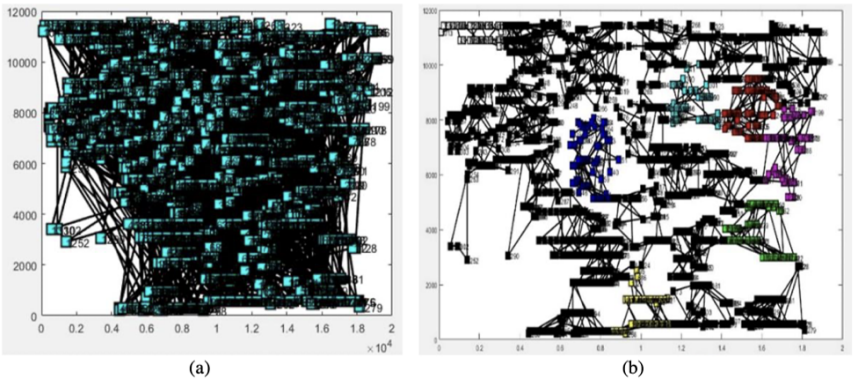


Figure 5.2.3: Figure a) Non-clustered WOA for Rl1323 Dataset ; Figure b) Clustered WOA for Rl1323 Dataset

5.3 The clustering Method for whale algorithm by Birch Algorithm

In another method, we clustered the dataset with BIRCH (Balance Iterative Reducing and Clustering using Hierarchies) and in the last step, we applied k-means to solve the TSP problem [81]. These are the steps [81]:
Step1: Set the parameters (see table.) for 20 iterations and 100 population numbers:
a: decreasing number from 2 to 0 coefficient such that A= [-a, a]
r: random number [0,1]
X: the whale's position
X*: The best solution
P: random number [0, 1] used for probability
b: a constant number for logarithmic shape
l: a value in [-1,1]
D: distance
N: city's length
Step 2: Specify K randomly or using the below equation for big tours among the cities, and the Branching factor:

$$Br - Factor = \frac{N}{5} \tag{5.3.1}$$

$$k = \sqrt{(\frac{N}{10})} \tag{5.3.2}$$

For the threshold:

$$S = 5 \times \sigma(\sigma(D)) \tag{5.3.3}$$

Step 3: Applying the Birch algorithm to cluster our data
Step 4: Applying the WOA algorithm for all of the found clusters i =1:K
Step 5: Find the location of the Agents

Step 6: Sorting by indexing
Step 7: Joining the clusters by finding the cities that are closer to the centroids of that cluster
Step 8: Repeat till joining all the clusters
The summarization of our article is shown in the form of Table 5.8 and Table 5.9.

Non-clustered approach Fitness	Ali535	Rat783	Pr1002
Avg	35903.55	1.3169e+05	4.0951e+06
Max	36400.23	1.3475e+05	4.1810e+06
Min	35333.53	1.2895e+05	3.9822e+06
Stdev.s	46913.33	1.4941e+03	46913.33
Non-clustered approach Timing	Ali535	Rat783	Pr1002
Avg	9.03	13.05	16.31
Max	9.13	13.21	16.50
Min	8.95	12.98	16.19
Stdev.s	0.04	0.06	0.07

Table 5.8: Non-clustered Whale Optimization Algorithm for Solving TSP Problem

The figures for these datasets (Ali535, Rat783, Pr1002) are illustrated for the two methods. The first three figures show the Non-clustered approach in Figures 5.3.1 till figure 5.3.3. Then we show them after the combination with the Birch algorithm, respectively (in figures 5.3.4 till figure 5.3.6).

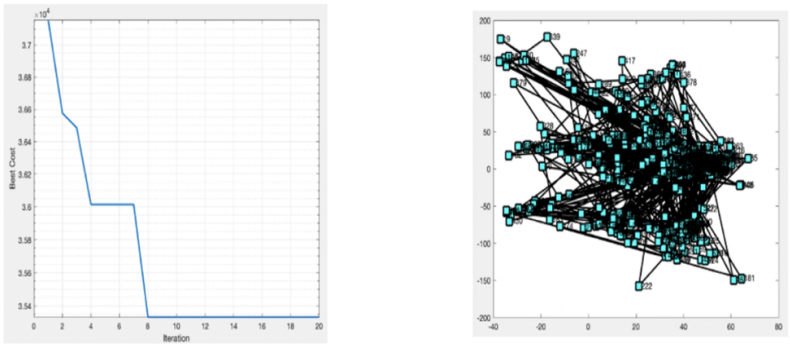


Figure 5.3.1: Effects of selecting the WOA algorithm to solve TSP for ali535: (a) the values of the cost function for ali535 and (b) Non-clustered approach

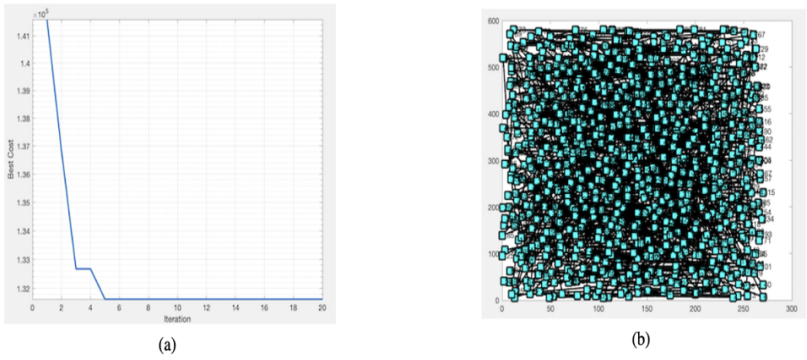


Figure 5.3.2: Effects of selecting the WOA algorithm to solve TSP for rat783: (a) the values of the cost function for rat783 and (b) Non-clustered approach

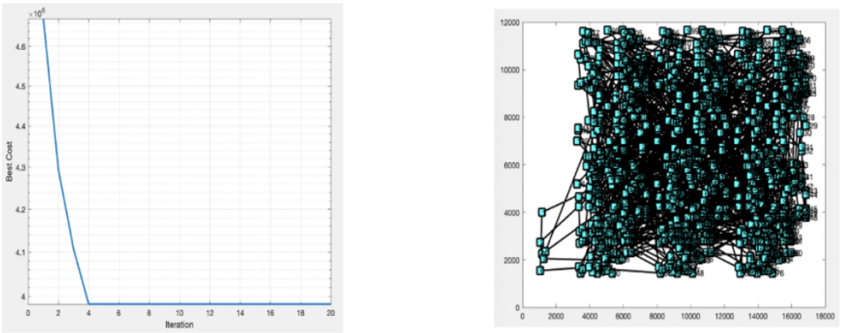


Figure 5.3.3: Effects of selecting the WOA algorithm to solve TSP for pr1002: (a) the values of the cost function for pr1002 and (b) Non-clustered approach

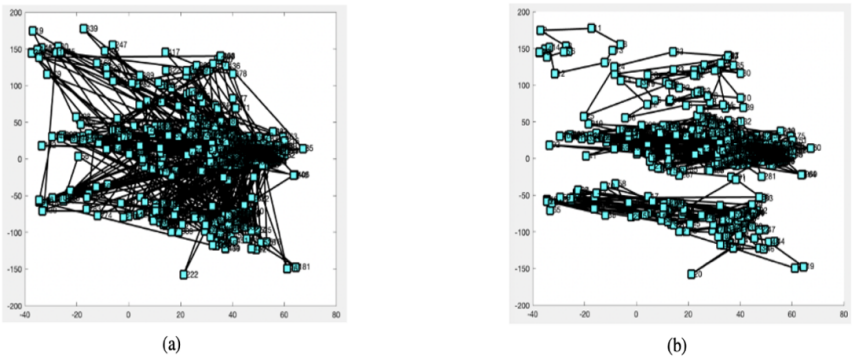


Figure 5.3.4: Effects of selecting the combined BIRCH and WOA algorithm to solve TSP for Ali535: (a) connected clusters and (b) the unjoined optimal clusters

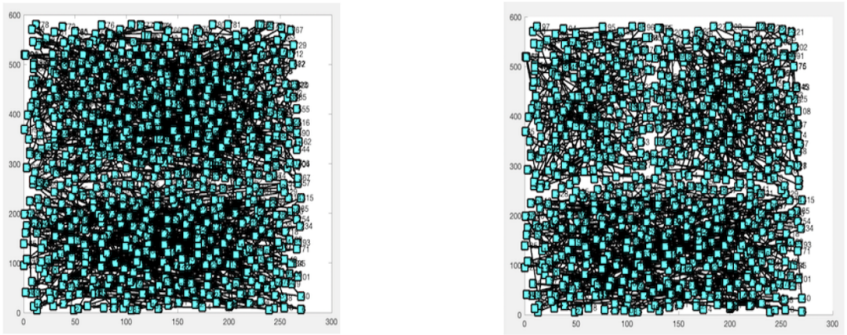


Figure 5.3.5: Effects of selecting the combined BIRCH and WOA algorithm to solve TSP for Rat783: (a) connected clusters and (b) the unjoined optimal clusters

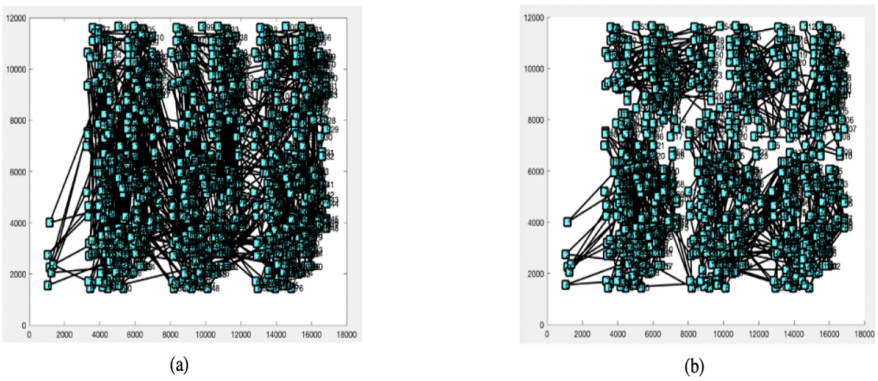


Figure 5.3.6: Effects of selecting the combined BIRCH and WOA algorithm to solve TSP for Pr1002: (a) connected clusters and (b) the unjoined optimal clusters

WBirch Fitness	Ali535	Rat783	Pr1002
Avg	36197.19	1.1329e+05	3.3316e+06
Max	36702.4	1.1434e+05	3.4310e+06
Min	35763.49	1.1233e+05	3.2389e+06
Stdev.s	49571.8325	642.4314	49571.83
WBirch Timing	Ali535	Rat783	Pr1002
Avg	4.15	5.18	6.48
Max	4.29	5.26	6.56
Min	4.11	5.15	6.46
Stdev.s	0.04	0.02	0.07

Table 5.9: Clusterized Whale Optimization Algorithm with Birch algorithm for Solving TSP Problem

5.4 New Scientific Achievements Of the Dissertation

Solving an NP-hard problem was the main subject matter. Here, we want to modify the introduced methods used in the two articles to improve fitness function in an acceptable time interval. We introduce three new methods. Two of them are our introduced methods for the thesis, and the third one becomes a practice for future works. The two new algorithms are tested over 318, 1002, and 1323 cities. The results of these two modified methods are close, but they can find more optimized clusters compared to the previous algorithms to achieve the most optimized solutions.

5.4.1 Thesis 1: The First New Proposed Method

In this method, after the initial clustering, the algorithm checks the number of cities. If the number exceeds the specified number (threshold), it applies clustering for that cluster again and applies the whale algorithm and k-means to find the shortest path. For example, we have five clusters(as in figure 5.4.1), and in each cluster, we have 3 or

2 cities, but in the first cluster, the number of cities is 9 (suppose 6 is a threshold), so, We cluster that again. After the second clustering, three subsets remain (a, b, c), and the total number of clusters becomes seven. This is an example of a possible sequence: a, b, c, Cluster 2, Cluster 3, Cluster 4, Cluster 5. It means After splitting cluster 1 into three clusters, cluster 1 won't exist anymore and we have a,b, and c as three individual clusters. The algorithm joins these six clusters together using the k-means and the Whale algorithm to find the optimal path. These seven clusters will be connected and the algorithm will check the path again. In each iteration, we will have new connections between these clusters, because the subset clusters will join to different outer clusters (cluster 2, cluster 3, Cluster 4, Cluster 5) to find the best path. This method repeats till meeting the stopping criteria which is finding the most optimized path. This method is called **the type one Cluster-Thresholding Method (CTM1)**. Figure 5.4.1 shows a small dataset but in reality, we can see clusters with 1323 cities and many subsets(sub-clusters).
The steps of CTM1 are as follows:
1-Parameter Settings (as in the previous section), T= Threshold (maximum number of cities)= 20, $k = \sqrt{\frac{N}{2}}$, iteration=20, Population size=100, a=2;
2- Initial Clustering;
3- Checking the number of the cities,

For found clusters $C_i = 1 : K$,

If the Number of the cities >T, Split that cluster;
4- Apply the proposed K-means and WOA algorithm (in my first article);
5-Find the position of the newly found **inner clusters(a, b, c)**;
6- Sorting by indexing;
7- Join inner clusters to outer Clusters;
8- Repeat the previous steps until you find the optimal path or meet the stopping criteria like till no cluster remains un-joined;

5.4.2 Thesis 2: The Second New Proposed Method

The first method and the second method are so close, but the main difference between them is that in the first model, the subset clusters could join to all the outer clusters (like cluster2 to cluster5) using the method of k-means and WOA, but in the second approach, the subset clusters can just join inside cluster1 (in the example). This method is called **the type two Cluster-Thresholding Method (CTM2)**.
Suppose we have a set of k clusters, and their size is defined by the number of cities. We assign a value for the threshold(T) which is the maximum number of cities for each cluster.
If the number of cities exceeds than T, our algorithm divides that cluster into smaller clusters. Then, the Whale algorithm and k- means start to look for the shortest path in each cluster. After sorting and indexing the cities, the final results(clusters) or solutions of each cluster must be connected. The final result will be the optimized path. If we have 9 cities (like in the previous example) and our Threshold is equal to 6, we cluster that again, and finally, we join the subset clusters together. This way the subset clusters become optimized and they can be connected inside of their main cluster(here, cluster1). In the previous figure, the algorithm splits cluster 1 into 3 subset clusters. Then, the three subsets (a, b, c) can be connected in a way that cluster 1 becomes optimized, like (a,b,c), (a,c,b), (b, c, a). The result of cluster 1 which is an optimized path would become connected to the other 4 clusters (cluster 2 to cluster 5). In this approach, cluster 1 still exists and one possible solution for this method can be the sequence of Cluster 1, Cluster 2, Cluster 3, Cluster 4, Cluster 5. It means we don't consider a, b, and c as individual clusters. We use them to optimize cluster 1 (inside optimization). They help the algorithm to find an optimal path but they don't have any connection with Cluster 2, Cluster 3, Cluster 4, Cluster 5, individually, and that's the Cluster 1(as an optimized cluster) that connects to the outer cluster. This

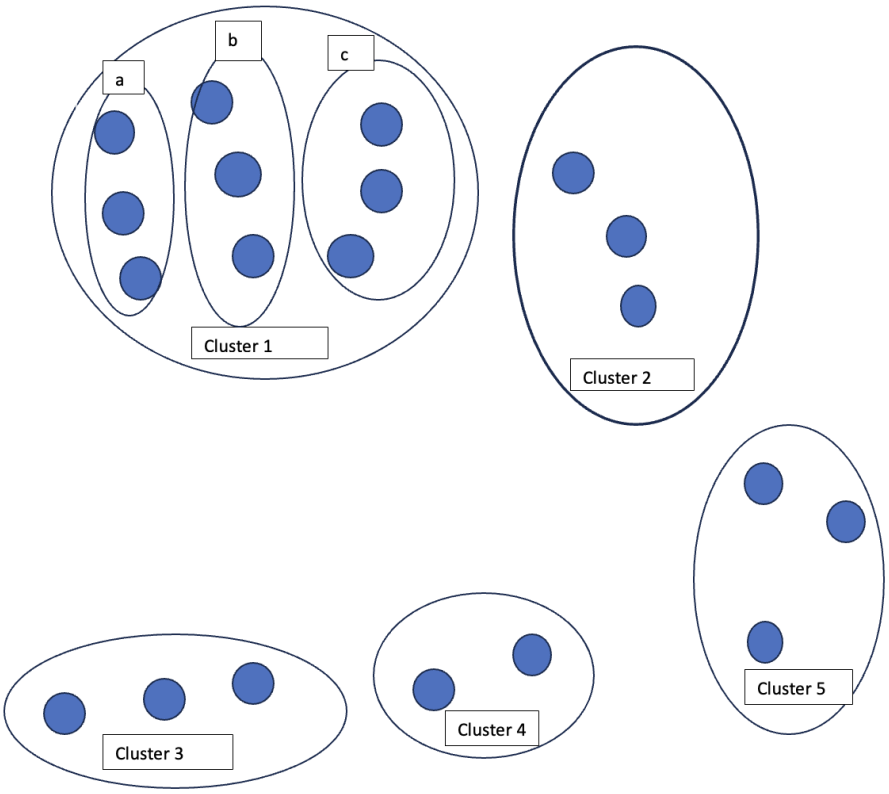


Figure 5.4.1: Example used for Showing the Procedure of the Two New Methods

way, the number of connections will be reduced as well.
These two algorithms are suitable for big data to solve an optimization problem (here TSP). In case we want to use them for a small dataset, the threshold value should be a small number, because if we assign the threshold (T=50) for a dataset like Eil51, we will only have just one cluster. For dataset Linhp318 we will have only 6 clusters, etc.

The steps of CTM2 are as follows:
1-Parameter Settings (as in the previous section), T= Threshold (maximum number of cities, here is n=20), Iteration=20, population size=100, a=2, $K = \sqrt{\frac{N}{2}}$,
2- Initial Clustering
3- Checking the number of the cities,

For found clusters $C_i = 1 : k$,

If the Number of the cities >T, Split that cluster;
4- Apply the proposed K-means and WOA Algorithm;
5-Find the position of the newly found **inner clusters(a, b, c)** and Connect them in an optimized path,
6- Sorting by indexing;
7- Join Cluster 1 to outer clusters;
8- Repeat the previous steps until you find the optimal path or meet the stopping criteria like till no cluster remains un-joined;

These two methods have close results, but they can find better answers compared to the simple version of the WOA algorithm for solving TSP. Tables 5.10 and 5.11 show the summary of the results obtained during clustering using these two approaches.

5.4.3 New Methods Evaluation

Table 5.10 shows the results for 318, 1002, and 1323 cities (from TSPLIB [82]) with the **CTM1 method** when n is 20:

CTM1 Fitness	Linhp318	Pr1002	Rl1323
Average	5.8023e+04	3.2570e+05	4.3395e+05
Max	5.9915e+04	3.6205e+05	4.5737e+05
Min	5.5231e+04	2.9857e+05	4.1532e+05
Stdev.s	1525.77	17340.43	11909.58
CTM1 Timing	Linhp318	Pr1002	Rl1323
Average	2.20668	7.04759	9.911245
Max	2.4493	7.1882	10.0954
Min	2.0761	6.8703	9.7484
Stdev.s	0.08	0.07	0.09

Table 5.10: The Fitness and Timing of the CTM1 Method for TSP

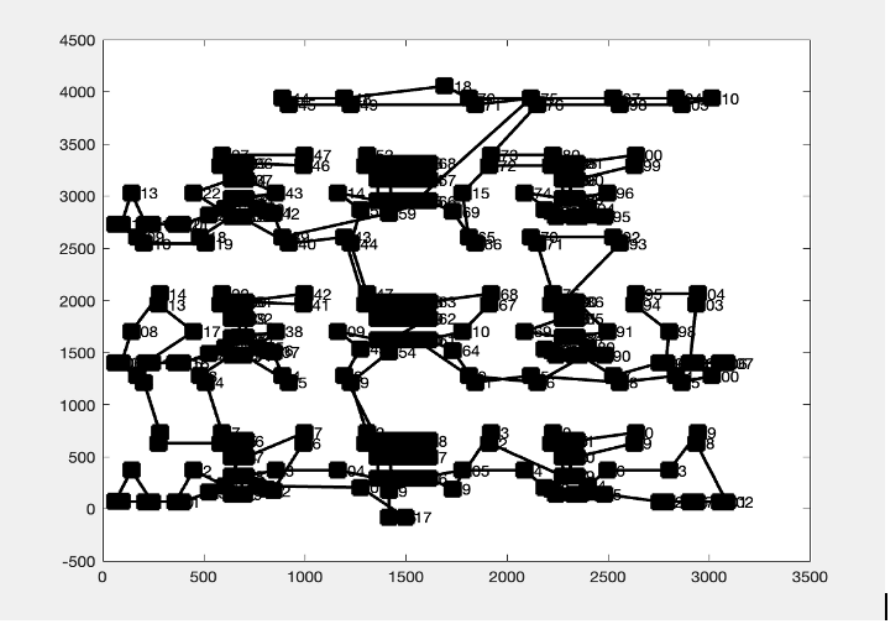
Table 5.11 shows the results for 318, 1002, and 1323 cities (from TSPLIB) with the **CTM2 method**, when n is 20:
Figures 5.4.2 to 5.4.4 show the way that the cities are connected in

CTM2 Fitness	Linhp318	Pr1002	Rl1323
Average	5.87518e+04	3.47833e+05	4.49069e+05
Max	6.2708e+04	3.6535e+05	4.7162e+05
Min	5.2073e+04	3.3138e+05	4.3298e+05
Stdev.s	2.4841e+03	8.079e +03	9.968e+03
CTM2 Timing	Linhp318	Pr1002	Rl1323
Average	2.181765	7.32871	10.146345
Max	2.2754	7.6146	10.4448
Min	2.0723	7.1189	10.0152
Stdev.s	0.06	0.12	0.10

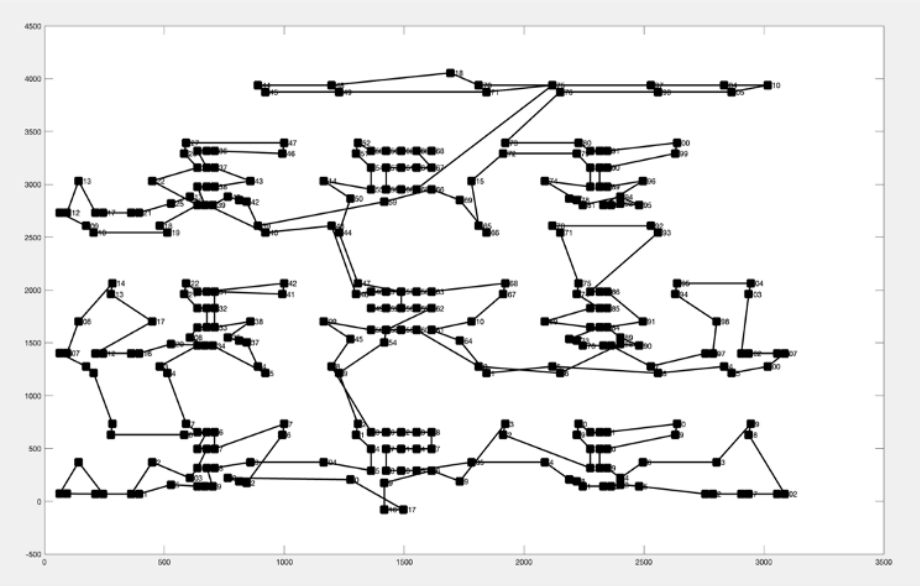
Table 5.11: The Fitness and Timing of the CTM2 Method for TSP

an optimized method for CTM1, and Figures 5.4.5 to 5.4.7 show the city's connection for CTM2.

Table 5.12 shows the test results of the Firefly algorithm(FA) for three datasets. Table 5.13 compares FA with some of the discussed

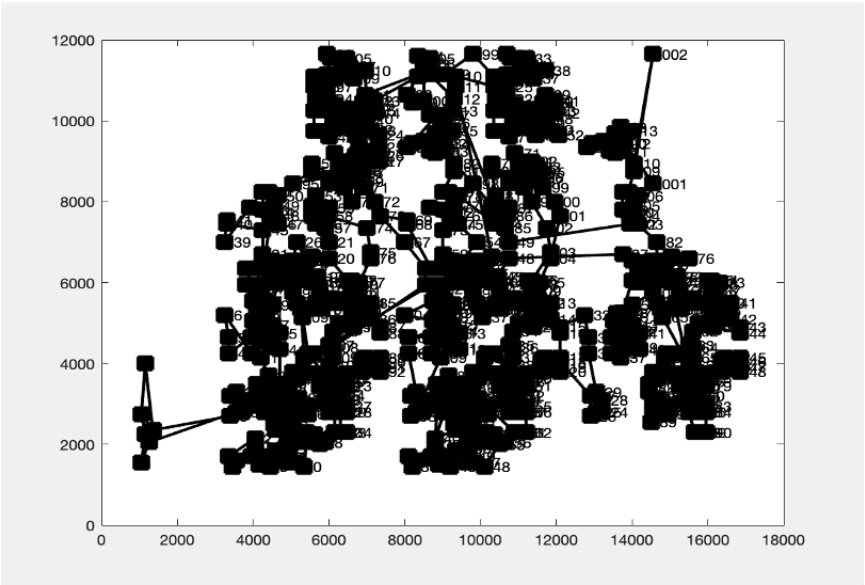


a) Dataset linhp318: illustration of TSP solver approach for the first approach

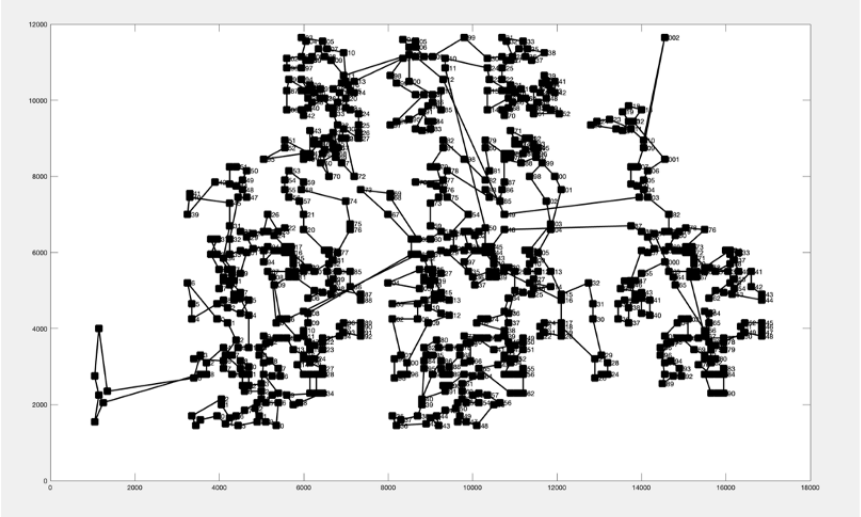


b) Dataset linhp318: illustration of TSP solver approach for the first approach

Figure 5.4.2

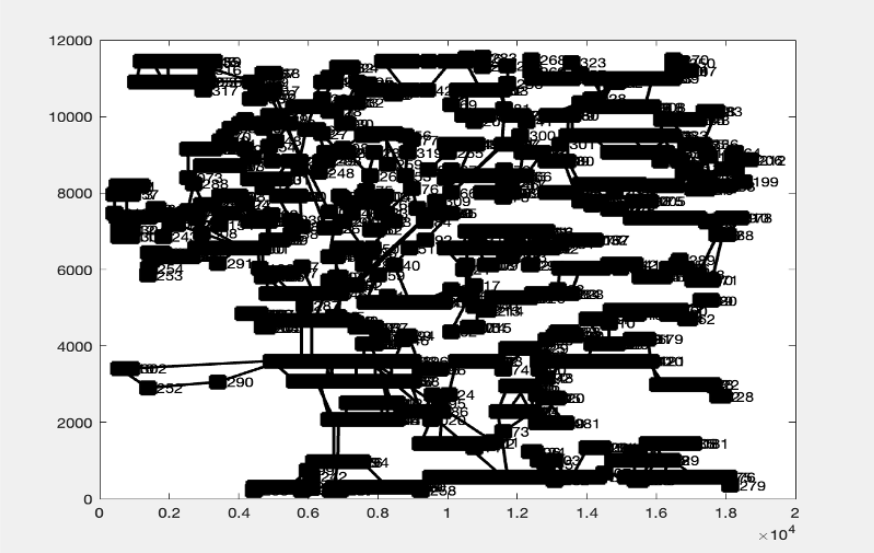


a) Dataset pr1002: illustration of TSP solver approach for the first approach

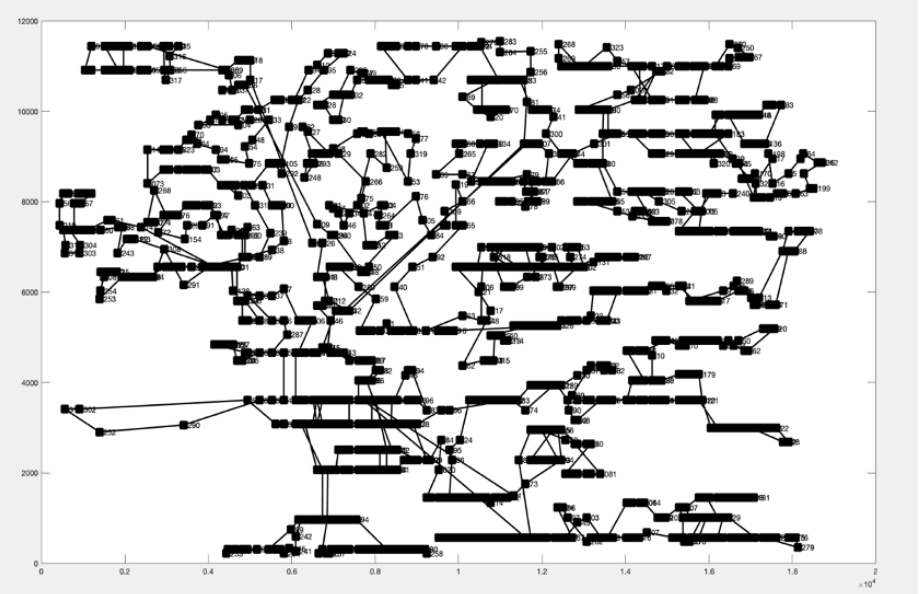


b) Dataset pr1002: Maximized illustration of TSP solver approach for the first approach

Figure 5.4.3: Figure 2

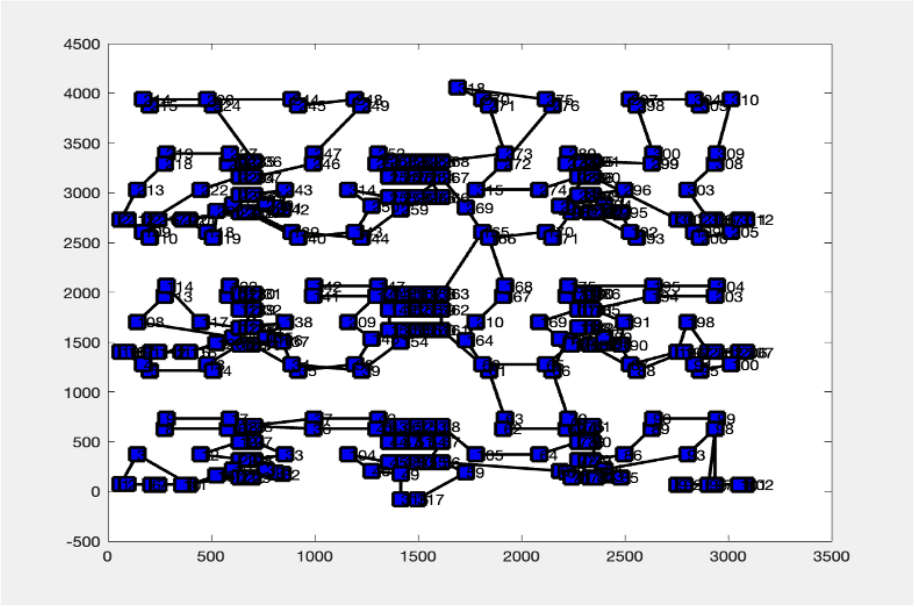


a) Dataset rl1323: illustration of TSP solver approach for the first approach

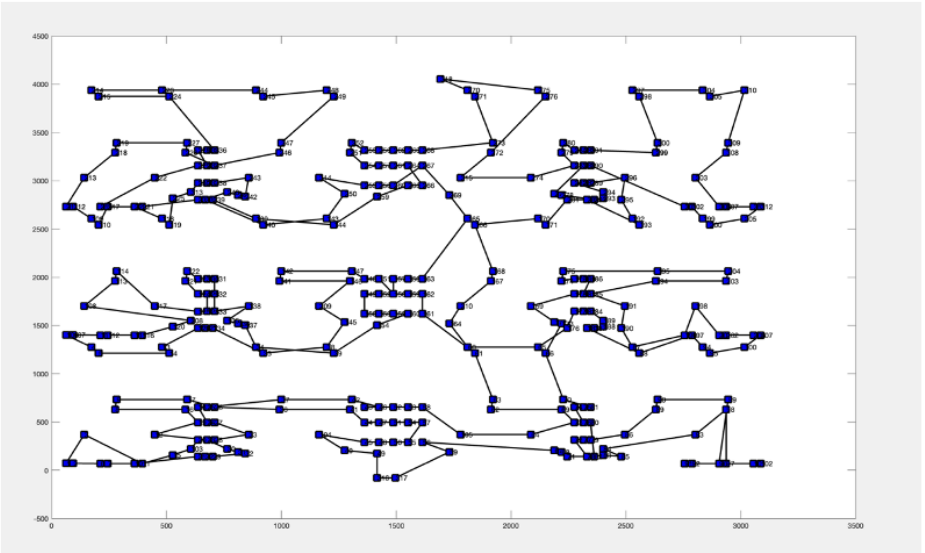


b) Dataset rl1323: Maximized illustration of TSP solver approach for the first approach

Figure 5.4.4

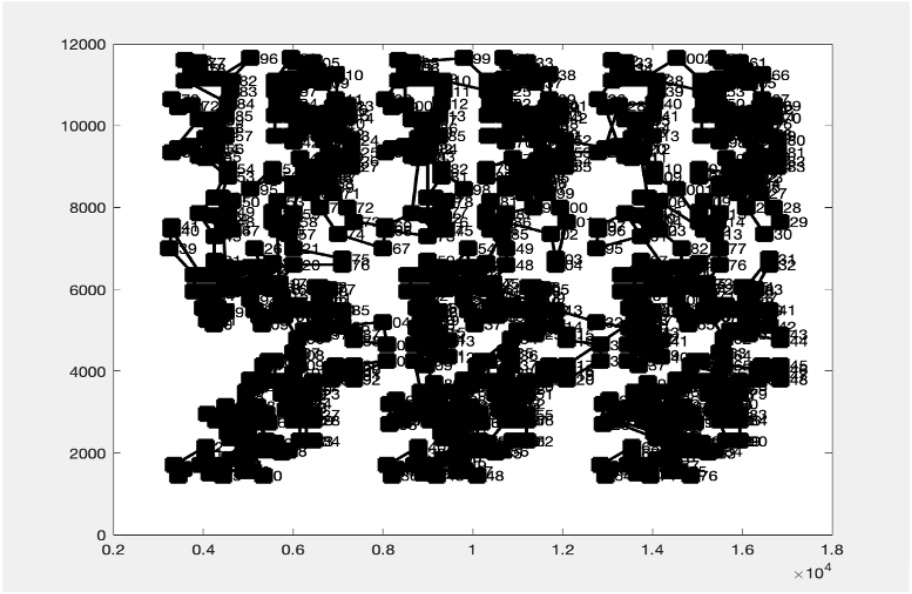


a) Dataset Linhp318: illustration of TSP solver approach for the second approach

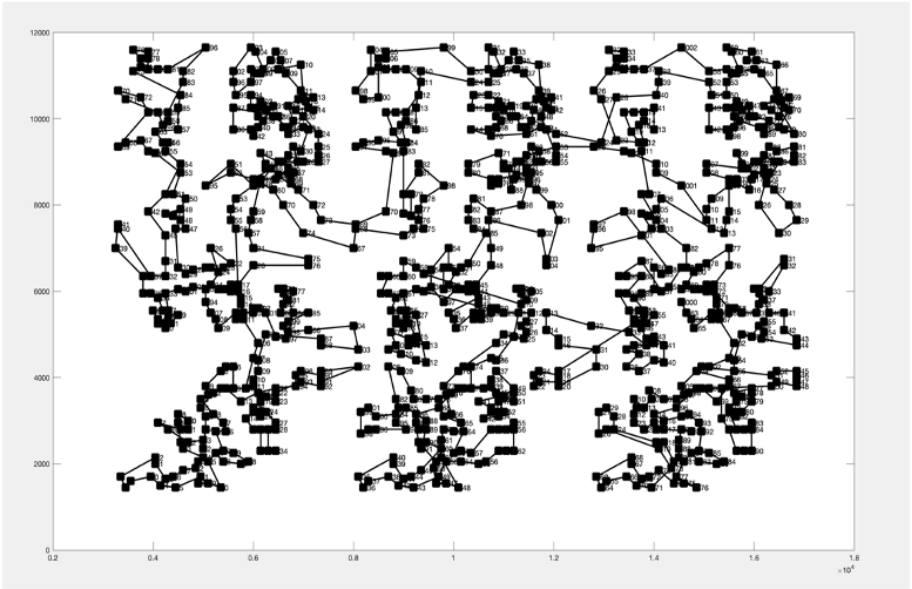


b) Dataset Linhp318: Maximized illustration of TSP solver approach for the second approach

Figure 5.4.5

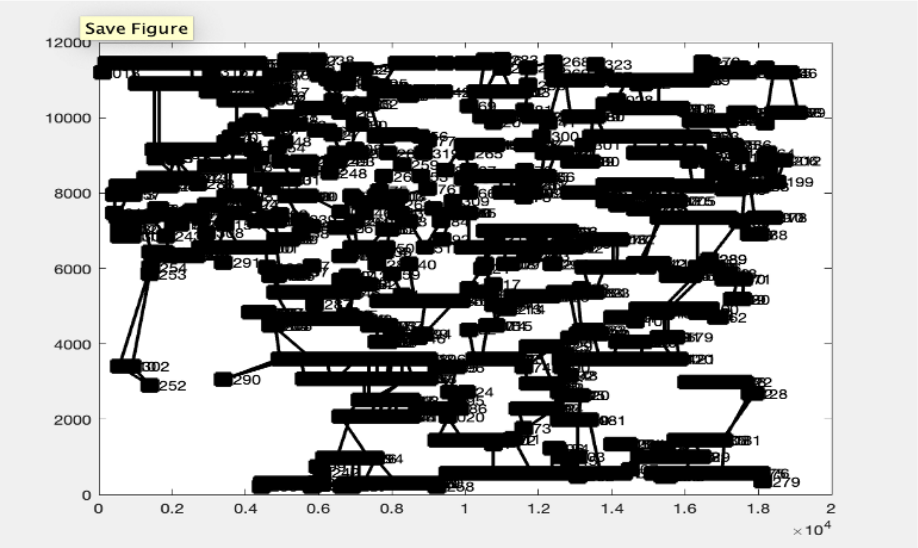


a) Dataset Pr1002: illustration of TSP solver approach for the second approach

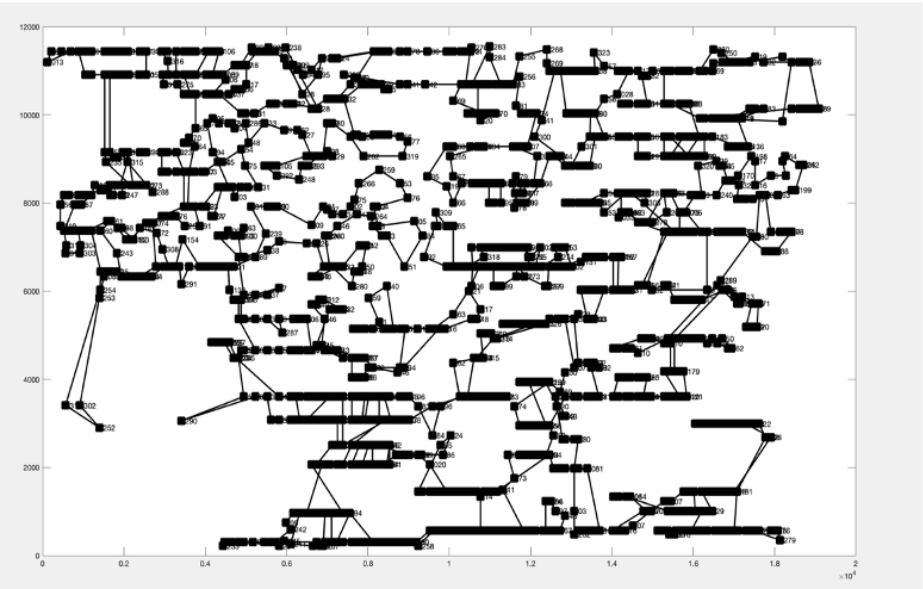


b) Dataset Pr1002: Maximized illustration of TSP solver approach for the second approach

Figure 5.4.6



Dataset Rl1323: illustration of TSP solver approach for the second approach



b) Dataset Rl1323: Maximized illustration of TSP solver approach for the second method

Figure 5.4.7

FA- TSP Fitness	eil51	KroB100	Linhp318
Avg	1296.2059	137481.3	359314,652
Max	1345.935	142758.8	371357,168
Min	1235.704	130426.8	341321,961
Stdev.s	32.27554	2826,941	7648,24076
FA- TSP Timing	eil51	KroB100	Linhp318
Avg	2,52988	3,57347	18,926205
Max	3,3725	6,3015	21,4923
Min	2,1979	2,9746	17,3522
Stdev.s	0,291414	0,705878	1,06364513

Table 5.12: Firefly algorithm Test Results for Solving TSP Problem

methods for the linhp318 dataset considering the minimum fitness value.
Table 5.14 and Table 5.15 show the minimum fitness and timing among the algorithms for the three datasets, respectively. The two other figures 5.4.8, and 5.4.9 are examples of TSP for some of the datasets based on a paper published in 2018 [83].

Data	Linhp318 Fitness	Linhp318 Timing
FA-TSP	341321,961	17,3522
WOA- TSP S	376614.9	13.642
WOA- TSP KMEANS	77273	4.6754
CT1	54489	5,5275

Table 5.13: The Comparison Between the Four Methods based on Their Fitness and Timing for TSP

Minimum Fitness	Linhp318	Pr1002	Rl1323
WOA- TSP	376614.9	3.9822e+06	8174234.322
WOA- TSP K-means	77273	6.7806e+05	1.14E+06
WOA+ Birch TSP	3.1516e+05	3.2389e+06	9.1221e+06
CTM1	5.5231e+04	2.9857e+05	4.1532e+05
CTM2	5.2073e+04	3.3138e+05	4.3298e+05

Table 5.14: Comparison Between the Fitness of the Five Methods for TSP

Minimum Timing	Linhp318	Pr1002	Rl1323
WOA- TSP	13.642	16.1910	51.669
WOA- TSP- K-means	4.6754	5.2557	20.345
WOA-Birch-TSP	2,9233	6.46	8.3500
CTM1	2.0761	6.8703	9.7484
CTM2	2.0723	7.1189	10.0152

Table 5.15: Comparison Between the Timing of the Five Methods for TSP

number of nodes	TSP no clustering	TSP with 2-level clustering	TSP with 3-level clustering	MTSP no clustering	MTSP with 2-level clustering
50	763	523	550	1328	1043
200	4358	1106	1202	6473	2385
400	10938	1647	1791	16820	4177
1000	36156	5801	3528	41052	6097

Figure 5.4.8: The length of the local optimum route [83]

number of nodes	TSP no clustering	TSP with 2-level clustering	TSP with 3-level clustering	MTSP no clustering	MTSP with 2-level clustering
50	5.9472	5.4596	9.2239	3.1367	6.2551
200	50.6509	20.6353	25.492	46.3972	21.1015
400	90.4167	4.384	6.1536	66.3395	4.6927
1000	729.451	12.1869	12.7243	555.1245	11.3891

Figure 5.4.9: The run time of optimization process (sec) [83]

5.5 Third Method: Future Works

The third method will be based on using the environment. As we know, Big Data needs big scale, so for these kinds of data, we need some more scalable algorithms. Our dataset can be divided into a pre-defined number, not based on similarity. We can apply k-means and WOA for each cluster. Then we combine the clusters. In the end, we have different sequences for the clusters that must be connected, and one of them which offers the shortest path will be the best. In Figure 5.5.1, we suppose that we have 4 clusters. For each cluster, we can apply K-means and WOA separately to find the best path. In the end, we join the clusters. We have different ways to connect them that is k!. For this example: 4!= 24. Some of the possible sequences are: $\{1-2-3-4\}$, $\{1-2-4-3\}$, $\{1-3-2-4\}$, $\{1-4-3-2\}$, and etc. One of these three ways of connecting the clusters will be the best path. We select the first-founded best path among the clusters.

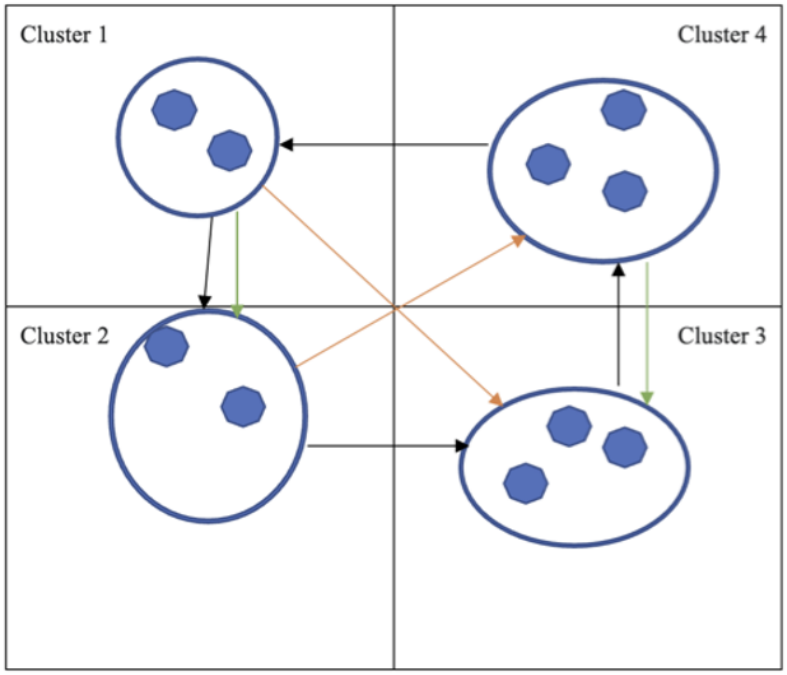


Figure 5.5.1: Third Approach for TSP

5.6 Some of the Source Codes for the Combined Whale Optimization algorithm

```
Here is the code sample for the non-clustered whale algorithm to solve TSP.
It is the main function of WOA:
clc;
close all;
```

```
clear;

% Generate data
% load eil76
%load eil51
% load pr76
% load ulysses16
load pr1002
dataset=pr1002;
Model=CreateModel(dataset);



X=[Model.x;
   Model.y];

XMax=max(dataset(1,:));
YMax=max(dataset(2,:));

Costfunction=@(tour) TourLength(tour,Model);
ub=10;
lb=-10;
nVar= Model.n ;            % Number of Decision Variables

VarSize=[1 nVar];
%% Whale parameters
nPop=100;
MaxIt=20;
a=2;

%% initialize step
tic
Empty_Whale.position=[];
Empty_Whale.fitt=[];
```

```
Whale=repmat(Empty_Whale,nPop,1);

bestsol.position=[];
bestsol.fitt=inf;
for i=1:nPop
    Whale(i).position=unifrnd(lb,ub,VarSize);
    Whale(i).fitt=Costfunction(Whale(i).position);

    if Whale(i).fitt <bestsol.fitt
        bestsol.position=Whale(i).position;
        bestsol.fitt=Whale(i).fitt;
    end

end

for it=1:MaxIt

      for   i=1:nPop
       for   j=1:nVar

           r1=rand;
           r2=rand;

           A=2*a*r1-a;
           c=2*r2;
           p=rand;


           if  p<0.5   % Not  Spiral
               if  abs(A)<=1
                   D=abs(c*bestsol.position(j)-Whale(i).position(j));
                   Whale(i).position(j)=bestsol.position(j)-A*D;
               else
                   K=randperm(nPop);
```

```
                    K(K==i)=[];
                    k=K(1);

                    D=abs(c*Whale(k).position(j)-Whale(i).position(j));
                    Whale(i).position(j)=Whale(k).position(j)-A*D;
                end
            else
                % Spiral
                b=1;
                l=unifrnd(-1,1);
                D=abs(bestsol.position(j)-Whale(i).position(j));
          Whale(i).position(j)=D*exp(b*l)*cos(2*pi*l)+ bestsol.position(j);
            end
        end
        % Bounded  Whale
        Whale(i).position=max(Whale(i).position,lb);
        Whale(i).position=min(Whale(i).position,ub);
        % Evaluation
        Whale(i).fitt=Costfunction(Whale(i).position);
       end
       % Update  a
      a=2-it*((2)/MaxIt);
   % Find  And  Store  And  Display  Best  Solution
   [~,mi]=min([Whale.fitt]);
   if   Whale(mi).fitt<bestsol.fitt
      bestsol=Whale(mi);
   end
   bestcost(it)=bestsol.fitt;
disp(['WOA Iteration:' num2str(it) '
Best  Fitness:'num2str(bestcost(it))]);
   pause(0.05);
   [~,inx]=sort(bestsol.position);
   PlotSolution(inx,Model)
```

```
end
ExeCTime=toc
figure;
semilogy(bestcost, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;

---------------------------------------------------
function model=CreateModel(data)



x=data(1,:);
y=data(2,:);

     n=numel(x);

    D=zeros(n,n);

    for i=1:n-1
        for j=i+1:n

            D(i,j)=sqrt((x(i)-x(j))^2+(y(i)-y(j))^2); Auclidean distance

            D(j,i)=D(i,j);

        end
    end

    model.n=n;
    model.x=x;
    model.y=y;
```

```
    model.D=D;
end
----------------------------------------
function L=TourLength(sol,Model)

    [~,ind]=sort(sol);


    tour=ind;
    n=numel(tour);

    tour=[tour tour(1)];

    L=0;
    for i=1:n
        L=L+Model.D(tour(i),tour(i+1));
    end

end
----------------------------------------
function PlotSolution(tour,model)

    tour=[tour tour(1)];
            plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[0 1 1]',...
        'LineWidth',2)
        for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
        end
```

```
end
```

Here is the combined k-means with the WOA algorithm:

```
load pr1002
dataset=pr1002;
Model=CreateModel(dataset);
X=[Model.x;
   Model.y];


k = round(sqrt(Model.n/2));


%% kmeans
tic
centroids=kseeds(X,k);

y = kmeans(X,centroids);

Cluster=[];
for i=1:k
    Cluster(i).Pos=find(y==i);
end

% figure(2)
% PlotClusters(Cluster,Model)
% xLimits = get(gca,'XLim');
% yLimits = get(gca,'YLim');
for i=1:k
```

```
 Cluster(i).Pos=WOA(Cluster(i).Pos,Model,Cluster,i);

end


result = FindCenroidindex(X,centroids);

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(:));
[row,col] = find(Y==minMatrix);

C1=Cluster(row).Pos;
C2=Cluster(col).Pos;
Cnew=C1;

out=[row col];
color=row;
% figure(3)
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

for i=1:k-1


C1=Cnew;
C2=Cluster(out(i+1)).Pos;



Z=CalcDisNodes(C1,C2,Model);

minMatrix = min(Z(:));
[row1,col1] = find(Z==minMatrix);
```

```
row1=row1(1);
col1=col1(1);
if find(C1==row1)==size(C1,2)
    Ncandidfirs1=C1(1);
else
    Ncandidfirs1=C1(find(C1==row1)+1);
end
if find(C1==row1)==1
    Ncandidfirs2=C1(end);
else
    Ncandidfirs2=C1(find(C1==row1)-1);

end


if find(C2==col1)==size(C2,2)
    Ncandidsecond1=C2(1);
else
    Ncandidsecond1=C2(find(C2==col1)+1);
end
if find(C2==col1)==1
    Ncandidsecond2=C2(end);
else
    Ncandidsecond2=C2(find(C2==col1)-1);
end

Z=CalcDisNodes([Ncandidfirs1 Ncandidfirs2],
[Ncandidsecond1 Ncandidsecond2],Model);
minMatrix = min(Z(:));
[row2,col2] = find(Z==minMatrix);
row2=row2(1);
col2=col2(1);
C1=Editsol(C1,row1,row2);
C2=Editsol(C2,col1,col2);
```

```
Cnew=[C1 flip(C2)];
% pause(0.5);
% if i==1
%     clf(figure(3));
% end
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

in=Subarray(1:k,out);
if isempty(in)
    break;
end

Cnew_cent=FindCenroidindex(X,kseeds([Model.x(Cnew);Model.y(Cnew)],1));
Other_cent=[];
for i=1:size(in,2)
  Other_cent(i)=FindCenroidindex
(X,kseeds([Model.x(Cluster(in(i)).Pos);Model.y(Cluster(in(i)).Pos)],1));
end

result=[Cnew_cent Other_cent];

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(1,:));
[row,col] = find(Y==minMatrix);

if(size(in,2)==1)
out=[out in];
else
out=[out in(col-1)];
end
end
ExeCTime=toc
```

```
TotalCost(Cnew,Model)
-----------------------------------------------------
here is FindCenroidindex():
function result = FindCenroidindex(X,centroids)
result=zeros(1,size(centroids,2));
for i=1:size(X,2)
for j=1:size(centroids,2)

   if X(1,i)==centroids(1,j) &&  X(2,i)==centroids(2,j)
      result(1,j)=i;
   end
end
end


end
-----------------------------------------------------------
function Y=CalcDisNodes(C1,C2,Model)

n=Model.n;
Y=zeros(n,n);

    for i=1:n
        for j=1:n

           Y(i,j)=inf;

        end
    end
    for i=1:size(C1,2)
        for j=1:size(C2,2)
          Y(C1(i),C2(j))=Model.D(C1(i),C2(j));
```

```
        end
    end

end
-------------------------------------------------------------
function Y = CalcDisCluster(x,Model)

n=numel(x);
Y=zeros(n,n);
    for i=1:n
        for j=1:n

           Y(i,j)=inf;

        end
    end



    for i=1:n-1
        for j=i+1:n

            Y(i,j)=Model.D(x(i),x(j));
            Y(j,i)=inf;



        end
    end
end
-------------------------------------------------------
function out = Subarray(Y,X)

out=Y;
```

```
for i=1:size(Y,2)
    for j=1:size(X,2)
        if Y(i)==X(j)
         out(i)=0;
        end
    end
end

 out(out==0)=[];

end
-----------------------------------------------------------
function L = TotalCost(tour,model)

    n=numel(tour);

    tour=[tour tour(1)];

    L=0;
    for i=1:n
        L=L+model.D(tour(i),tour(i+1));
    end

end
-----------------------------------------------------------
Here is CTM1 Method:
clc;
close all;
clear;

% Generate data

load rl1323
dataset=rl1323;
```

```
Model=CreateModel(dataset);
X=[Model.x;
   Model.y];


k = round(sqrt(Model.n/2));

%% kmeans
tic
centroids=kseeds(X,k);

y = kmeans(X,centroids);

Cluster=[];
for i=1:k
    Cluster(i).Pos=find(y==i);
end

count=1;
TempCluster=[];
for i=1:k
    if size(Cluster(i).Pos,2)>20 % Thresholding

    DC=DivideCluster(Cluster(i).Pos,Model);
    for j=1:size(DC,2)
    TempCluster(count).Pos=DC(j).Pos;
    count=count+1;
    end
    else
    TempCluster(count).Pos=Cluster(i).Pos;
    count=count+1;
    end

end
```

```
if size(Cluster,2)~=size(TempCluster,2)
Cluster=TempCluster;
k=size(Cluster,2);

for i=1:k
centroids(:,i)=kseeds([Model.x(Cluster(i).Pos);Model.y(Cluster(i).Pos)],1);
end
end
%
% figure(2)
% PlotClusters(Cluster,Model)
% xLimits = get(gca,'XLim');
% yLimits = get(gca,'YLim');
for i=1:k
 Cluster(i).Pos=WOA(Cluster(i).Pos,Model,Cluster,i);

end


result = FindCenroidindex(X,centroids);

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(:));
[row,col] = find(Y==minMatrix);

C1=Cluster(row).Pos;
C2=Cluster(col).Pos;
Cnew=C1;

out=[row col];
color=row;
% figure(3)
% PlotSolution(Cnew,Model,color)
```

```
% xlim(xLimits)
% ylim(yLimits)

for i=1:k-1


C1=Cnew;
C2=Cluster(out(i+1)).Pos;



Z=CalcDisNodes(C1,C2,Model);

minMatrix = min(Z(:));
[row1,col1] = find(Z==minMatrix);
row1=row1(1);
col1=col1(1);
if find(C1==row1)==size(C1,2)
    Ncandidfirs1=C1(1);
else
    Ncandidfirs1=C1(find(C1==row1)+1);
end
if find(C1==row1)==1
    Ncandidfirs2=C1(end);
else
    Ncandidfirs2=C1(find(C1==row1)-1);

end


if find(C2==col1)==size(C2,2)
    Ncandidsecond1=C2(1);
else
    Ncandidsecond1=C2(find(C2==col1)+1);
```

```
end
if find(C2==col1)==1
    Ncandidsecond2=C2(end);
else
    Ncandidsecond2=C2(find(C2==col1)-1);
end

Z=CalcDisNodes([Ncandidfirs1 Ncandidfirs2],
[Ncandidsecond1 Ncandidsecond2],Model);
minMatrix = min(Z(:));
[row2,col2] = find(Z==minMatrix);
row2=row2(1);
col2=col2(1);
C1=Editsol(C1,row1,row2);
C2=Editsol(C2,col1,col2);
Cnew=[C1 flip(C2)];
% pause(0.5);
% if i==1
%     clf(figure(3));
% end
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

in=Subarray(1:k,out);
if isempty(in)
    break;
end

Cnew_cent=FindCenroidindex(X,kseeds([Model.x(Cnew);Model.y(Cnew)],1));
Other_cent=[];
for i=1:size(in,2)
  Other_cent(i)=FindCenroidindex
(X,kseeds([Model.x(Cluster(in(i)).Pos);Model.y(Cluster(in(i)).Pos)],1));
```

```
end

result=[Cnew_cent Other_cent];

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(1,:));
[row,col] = find(Y==minMatrix);

if(size(in,2)==1)
out=[out in];
else
out=[out in(col-1)];
end

end
ExeCTime=toc


TotalCost(Cnew,Model)
-----------------------------------------------------------------
function Cluster = DivideCluster(C,Model)
n=size(C,2);
k = round(sqrt(n/2));
X=[Model.x(C);Model.y(C)];
centroids=kseeds(X,k);
y = kmeans(X,centroids);

for i=1:k
    inx=(y==i);
    Cluster(i).Pos=C(inx);
end

end
-----------------------------------------------------------------
```

```
function PlotClusters(Cluster,Model)
clf(figure(2));
for i=1:size(Cluster,2)
 PlotSolution(Cluster(i).Pos,Model,i);
 hold on;
end
end
-----------------------------------------------------------------
function PlotSolution(tour,model,cl)

    tour=[tour tour(1)];
    switch cl
        case 1
            plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[0 0 0]',...
        'LineWidth',2)
        for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
        end

        case 2
            plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[0 0 1]',...
        'LineWidth',2)
        for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
        end

         case 3
             plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[0 1 0]',...
```

```
'LineWidth',2)
    for i=1:size(tour,2)
text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
    end

 case 4
     plot(model.x(tour),model.y(tour),'k-s',...
'MarkerSize',12,...
'MarkerFaceColor','[0 1 1]',...
'LineWidth',2)
    for i=1:size(tour,2)
text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
end
 case 5
     plot(model.x(tour),model.y(tour),'k-s',...
'MarkerSize',12,...
'MarkerFaceColor','[1 0 0]',...
'LineWidth',2)
    for i=1:size(tour,2)
text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
end
 case 6
     plot(model.x(tour),model.y(tour),'k-s',...
'MarkerSize',12,...
'MarkerFaceColor','[1 0 1]',...
'LineWidth',2)
    for i=1:size(tour,2)
text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
end
 case 7
     plot(model.x(tour),model.y(tour),'k-s',...
'MarkerSize',12,...
'MarkerFaceColor','[1 1 0]',...
'LineWidth',2)
```

```
      for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
        end
             case 8
             plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[1 1 1]',...
        'LineWidth',2)
      for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
      end

        otherwise
      plot(model.x(tour),model.y(tour),'k-s',...
        'MarkerSize',12,...
        'MarkerFaceColor','[0 0 0]',...
        'LineWidth',2)
      for i=1:size(tour,2)
        text(model.x(tour(i))+1,model.y(tour(i))+1,num2str(tour(i)))
      end


    end

end
-------------------------------------------------------------------
function L = TotalCost(tour,model)

    n=numel(tour);

    tour=[tour tour(1)];

    L=0;
    for i=1:n
```

```
        L=L+model.D(tour(i),tour(i+1));
    end
end
```

```
The first Approach CTM1:
clc;
close all;
clear;

% Generate data
%load eil51;
% load eil76
% load pr76
% load ulysses16
% load eil51;
load rl1323
%load pr1002
%load linhp318
dataset=rl1323;
Model=CreateModel(dataset);
X=[Model.x;
   Model.y];


k = round(sqrt(Model.n/2));

%% kmeans
tic
centroids=kseeds(X,k);

y = kmeans(X,centroids);

Cluster=[];
for i=1:k
```

```
    Cluster(i).Pos=find(y==i);
end

count=1;
TempCluster=[];
for i=1:k
    if size(Cluster(i).Pos,2)>20

    DC=DivideCluster(Cluster(i).Pos,Model);
    for j=1:size(DC,2)
    TempCluster(count).Pos=DC(j).Pos;
    count=count+1;
    end
    else
    TempCluster(count).Pos=Cluster(i).Pos;
    count=count+1;
    end

end

if size(Cluster,2)~=size(TempCluster,2)
Cluster=TempCluster;
k=size(Cluster,2);

for i=1:k
centroids(:,i)=kseeds([Model.x(Cluster(i).Pos);Model.y(Cluster(i).Pos)],1);
end
end
%
% figure(2)
% PlotClusters(Cluster,Model)
% xLimits = get(gca,'XLim');
% yLimits = get(gca,'YLim');
for i=1:k
```

```
 Cluster(i).Pos=WOA(Cluster(i).Pos,Model,Cluster,i);

end


result = FindCenroidindex(X,centroids);

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(:));
[row,col] = find(Y==minMatrix);

C1=Cluster(row).Pos;
C2=Cluster(col).Pos;
Cnew=C1;

out=[row col];
color=row;
% figure(3)
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

for i=1:k-1


C1=Cnew;
C2=Cluster(out(i+1)).Pos;



Z=CalcDisNodes(C1,C2,Model);

minMatrix = min(Z(:));
[row1,col1] = find(Z==minMatrix);
```

```
row1=row1(1);
col1=col1(1);
if find(C1==row1)==size(C1,2)
    Ncandidfirs1=C1(1);
else
    Ncandidfirs1=C1(find(C1==row1)+1);
end
if find(C1==row1)==1
    Ncandidfirs2=C1(end);
else
    Ncandidfirs2=C1(find(C1==row1)-1);

end


if find(C2==col1)==size(C2,2)
    Ncandidsecond1=C2(1);
else
    Ncandidsecond1=C2(find(C2==col1)+1);
end
if find(C2==col1)==1
    Ncandidsecond2=C2(end);
else
    Ncandidsecond2=C2(find(C2==col1)-1);
end

Z=CalcDisNodes([Ncandidfirs1 Ncandidfirs2],
[Ncandidsecond1 Ncandidsecond2],Model);
minMatrix = min(Z(:));
[row2,col2] = find(Z==minMatrix);
row2=row2(1);
col2=col2(1);
C1=Editsol(C1,row1,row2);
C2=Editsol(C2,col1,col2);
```

```
Cnew=[C1 flip(C2)];
% pause(0.5);
% if i==1
%     clf(figure(3));
% end
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

in=Subarray(1:k,out);
if isempty(in)
    break;
end

Cnew_cent=FindCenroidindex(X,kseeds([Model.x(Cnew);Model.y(Cnew)],1));
Other_cent=[];
for i=1:size(in,2)
Other_cent(i)=FindCenroidindex(X,kseeds([Model.x(Cluster(in(i)).Pos);
Model.y(Cluster(in(i)).Pos)],1));
end

result=[Cnew_cent Other_cent];

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(1,:));
[row,col] = find(Y==minMatrix);

if(size(in,2)==1)
out=[out in];
else
out=[out in(col-1)];
end

end
```

```
ExeCTime=toc


TotalCost(Cnew,Model)

---------------

Function for CTM1:

function Cluster = DivideCluster(C,Model)
n=size(C,2);
k = round(sqrt(n/2));
X=[Model.x(C);Model.y(C)];
centroids=kseeds(X,k);
y = kmeans(X,centroids);

for i=1:k
    inx=(y==i);
    Cluster(i).Pos=C(inx);
end

end









-----------------------------------
function BestTour=WOA(City,model,Cluster,NC)
```

```
Costfunction=@(tour) TourLength(tour,model,City);
ub=10;
lb=-10;
nVar=numel(City);               % Number of Decision Variables

VarSize=[1 nVar];
%% Whale parametrs
nPop=100;
MaxIt=20;
a=2;

%% initialaze step

Empty_Whale.position=[];
Empty_Whale.fitt=[];
Whale=repmat(Empty_Whale,nPop,1);

bestsol.position=[];
bestsol.fitt=inf;
for i=1:nPop
    Whale(i).position=unifrnd(lb,ub,VarSize);
    Whale(i).fitt=Costfunction(Whale(i).position);

    if Whale(i).fitt <bestsol.fitt
        bestsol.position=Whale(i).position;
        bestsol.fitt=Whale(i).fitt;
    end

end

for it=1:MaxIt

      for   i=1:nPop
       for   j=1:nVar
```

```
    r1=rand;
    r2=rand;

    A=2*a*r1-a;
    c=2*r2;
    p=rand;


    if  p<0.5   % Not  Spiral
        if  abs(A)<=1
            D=abs(c*bestsol.position(j)-Whale(i).position(j));
            Whale(i).position(j)=bestsol.position(j)-A*D;
        else
            K=randperm(nPop);
            K(K==i)=[];
            k=K(1);

            D=abs(c*Whale(k).position(j)-Whale(i).position(j));
            Whale(i).position(j)=Whale(k).position(j)-A*D;
        end
    else
        % Spiral
        b=1;
        l=unifrnd(-1,1);
        D=abs(bestsol.position(j)-Whale(i).position(j));
        Whale(i).position(j)=
        D*exp(b*l)*cos(2*pi*l)+bestsol.position(j);
    end
end
% Bounded  Whale
Whale(i).position=max(Whale(i).position,lb);
Whale(i).position=min(Whale(i).position,ub);
% Evaluation
```

```
        Whale(i).fitt=Costfunction(Whale(i).position);
       end
       % Update  a
      a=2-it*((2)/MaxIt);
    % Find  And  Store  And  Display  Best  Solution
    [~,mi]=min([Whale.fitt]);
    if   Whale(mi).fitt<bestsol.fitt
       bestsol=Whale(mi);
    end
    bestcost(it)=bestsol.fitt;
% disp(['Optimize Cluster ' num2str(NC) '-->
Iteration:' num2str(it) '   Best  Fitness:' num2str(bestcost(it))]);
%    pause(0.05);
%    [~,inx]=sort(bestsol.position);
%     Cluster(NC).Pos=City(inx);
%     PlotClusters(Cluster,model)

end

[~,inx]=sort(bestsol.position);
BestTour=City(inx);
end


------------------
function  sol=Editsol(C,ind1,ind2)

n=size(C,2);
first=find(C==ind1);
endf=find(C==ind2);

 if first+1==endf
   sol(1)=C(first);
   i=n-1;
```

```
   count=2;
while (i>0)
    first=first-1;
    if first==0
        first=n;
    end
    sol(count)=C(first);
    i=i-1;
    count=count+1;


end
 else
sol(1)=C(first);
i=2;
while (i<n)
    first=first+1;
    if mod(first,n+1)==0
        first=1;
    end
    if C(first)~=C(endf)
    sol(i)=C(first);
    i=i+1;
    end

end
sol(n)=C(endf);
end
% end
---------------------
For the second approach CTM 2:
clc;
close all;
clear;
```

```
% Generate data
%load eil51;
% load eil76
% load pr76
% load ulysses16
% load eil51;
%load pr1002
%load rl1323
load linhp318
dataset=linhp318;
Model=CreateModel(dataset);
X=[Model.x;
   Model.y];


k = round(sqrt(Model.n/2));

MaxCity=20;

%% kmeans
tic
centroids=kseeds(X,k);

y = kmeans(X,centroids);

Cluster=[];
for i=1:k
    Cluster(i).Pos=find(y==i);
end

count=1;
TempCluster=[];
for i=1:k
```

```
    if size(Cluster(i).Pos,2)>=MaxCity

    DC=ResolveCluster(Cluster(i).Pos,Model);

    TempCluster(count).Pos=DC;

    centroids(:,i)=kseeds([Model.x(Cluster(i).Pos);Model.y(Cluster(i).Pos)],1);
    count=count+1;

    else
    TempCluster(count).Pos=Cluster(i).Pos;
    count=count+1;
    end

end

Cluster=TempCluster;

%
% figure(2)
% PlotClusters(Cluster,Model)
% xLimits = get(gca,'XLim');
% yLimits = get(gca,'YLim');
for i=1:k
    if size(Cluster(i).Pos,2)<MaxCity
     Cluster(i).Pos=WOA(Cluster(i).Pos,Model,Cluster,i);
    end

end


result = FindCenroidindex(X,centroids);

Y=CalcDisCluster(result,Model);
```

```
minMatrix = min(Y(:));
[row,col] = find(Y==minMatrix);

C1=Cluster(row).Pos;
C2=Cluster(col).Pos;
Cnew=C1;

out=[row col];
color=row;
% figure(3)
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

for i=1:k-1


C1=Cnew;
C2=Cluster(out(i+1)).Pos;



Z=CalcDisNodes(C1,C2,Model);

minMatrix = min(Z(:));
[row1,col1] = find(Z==minMatrix);
row1=row1(1);
col1=col1(1);
if find(C1==row1)==size(C1,2)
    Ncandidfirs1=C1(1);
else
    Ncandidfirs1=C1(find(C1==row1)+1);
end
if find(C1==row1)==1
```

```
    Ncandidfirs2=C1(end);
else
    Ncandidfirs2=C1(find(C1==row1)-1);

end


if find(C2==col1)==size(C2,2)
    Ncandidsecond1=C2(1);
else
    Ncandidsecond1=C2(find(C2==col1)+1);
end
if find(C2==col1)==1
    Ncandidsecond2=C2(end);
else
    Ncandidsecond2=C2(find(C2==col1)-1);
end

Z=CalcDisNodes([Ncandidfirs1 Ncandidfirs2],[Ncandidsecond1 Ncandidsecond2],Model
minMatrix = min(Z(:));
[row2,col2] = find(Z==minMatrix);
row2=row2(1);
col2=col2(1);
C1=Editsol(C1,row1,row2);
C2=Editsol(C2,col1,col2);
Cnew=[C1 flip(C2)];
% pause(0.5);
% if i==1
%     clf(figure(3));
% end
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)
```

```
in=Subarray(1:k,out);
if isempty(in)
    break;
end

Cnew_cent=FindCenroidindex(X,kseeds([Model.x(Cnew);Model.y(Cnew)],1));
Other_cent=[];
for i=1:size(in,2)
Other_cent(i)=FindCenroidindex(X,kseeds([Model.x(Cluster(in(i)).Pos);
Model.y(Cluster(in(i)).Pos)],1));
end

result=[Cnew_cent Other_cent];

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(1,:));
[row,col] = find(Y==minMatrix);

if(size(in,2)==1)
out=[out in];
else
out=[out in(col-1)];
end

end
ExeCTime=toc


TotalCost(Cnew,Model)

-----------------------------------------------
Function FOR CTM2:
function Cnew = ResolveCluster(C,Model)
n=size(C,2);
```

```
k = round(sqrt(n/2));
X=[Model.x(C);Model.y(C)];
centroids=kseeds(X,k);
y = kmeans(X,centroids);

for i=1:k
    inx=(y==i);
    Cluster(i).Pos=C(inx);
end

for i=1:k
 Cluster(i).Pos=WOA(Cluster(i).Pos,Model,Cluster,i);

end

result = FindCenroidindex(X,centroids);

Y=CalcDisCluster(result,Model);
minMatrix = min(Y(:));
[row,col] = find(Y==minMatrix);

C1=Cluster(row).Pos;
C2=Cluster(col).Pos;
Cnew=C1;

out=[row col];
color=row;
% figure(3)
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

for i=1:k-1
```

```
C1=Cnew;
C2=Cluster(out(i+1)).Pos;



Z=CalcDisNodes(C1,C2,Model);

minMatrix = min(Z(:));
[row1,col1] = find(Z==minMatrix);
row1=row1(1);
col1=col1(1);
if find(C1==row1)==size(C1,2)
    Ncandidfirs1=C1(1);
else
    Ncandidfirs1=C1(find(C1==row1)+1);
end
if find(C1==row1)==1
    Ncandidfirs2=C1(end);
else
    Ncandidfirs2=C1(find(C1==row1)-1);

end


if find(C2==col1)==size(C2,2)
    Ncandidsecond1=C2(1);
else
    Ncandidsecond1=C2(find(C2==col1)+1);
end
if find(C2==col1)==1
    Ncandidsecond2=C2(end);
else
    Ncandidsecond2=C2(find(C2==col1)-1);
```

```
end

Z=CalcDisNodes([Ncandidfirs1 Ncandidfirs2],
[Ncandidsecond1 Ncandidsecond2],Model);
minMatrix = min(Z(:));
[row2,col2] = find(Z==minMatrix);
row2=row2(1);
col2=col2(1);
C1=Editsol(C1,row1,row2);
C2=Editsol(C2,col1,col2);
Cnew=[C1 flip(C2)];
% pause(0.5);
% if i==1
%     clf(figure(3));
% end
% PlotSolution(Cnew,Model,color)
% xlim(xLimits)
% ylim(yLimits)

in=Subarray(1:k,out);
if isempty(in)
    break;
end

Cnew_cent=FindCenroidindex(X,kseeds([Model.x(Cnew);Model.y(Cnew)],1));
Other_cent=[];
for i=1:size(in,2)
Other_cent(i)=FindCenroidindex(X,kseeds([Model.x(Cluster(in(i)).Pos);
Model.y(Cluster(in(i)).Pos)],1));
end

result=[Cnew_cent Other_cent];

Y=CalcDisCluster(result,Model);
```

```
minMatrix = min(Y(1,:));
[row,col] = find(Y==minMatrix);

if(size(in,2)==1)
out=[out in];
else
out=[out in(col-1)];
end

end



end
```

Chapter 6

Main Statements and Conclusion of the dissertation

Based on the search results, in the new methods which are the improved versions, the algorithm can be converged to the best answer that is the shortest path in less time. In terms of the cost function, the two new methods also have a low-cost function. So, besides of hybrid WOA algorithm, these two new methods can also be considered as TSP solvers that are good for data with higher scales.
Hypothesis 1: All different sample datasets were electronically collected for all of the algorithms and methods [82]– and by more research works – more clustering algorithms or methods can be applied for example Density-based or grid-based clustering algorithms.
Hypothesis 2: With the help of data analyses we intended to prove the effectiveness of the methods in the statistical sense. Among the statistical measurements, the average and minimum are considered as benchmarks to check their efficiency.
The new methods that are tested on more than 1000 cities have the main characteristic that reduces the complexity of the problem.
We tested our algorithm many times to check the performance of the

new hybrid methods. The results show that these models are better than the original version (simple version) of WOA, especially for large-scale traveling salesman problems. This work can be expanded by combining it with other clustering methods or other bio-inspired algorithms like artificial Flora, firefly, etc.

Bibliography

[1] Osiakwan, C. N., & Akl, S. G. (1990, December). The maximum weight perfect matching problem for complete weighted graphs is in pc. In Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing 1990 (pp. 880-887). IEEE.

[2] Han, J., Kamber, M., & Pei, J. (2012). Data mining concepts and techniques third edition. University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University.

[3] Alhalabi, W., Kitanneh, O., Alharbi, A. et al. Efficient solution for finding Hamilton cycles in undirected graphs. SpringerPlus 5, 1192 (2016). https://doi.org/10.1186/s40064-016-2746-8

[4] Doerr, B., & Neumann, F. (Eds.). (2019). Theory of evolutionary computation: Recent developments in discrete optimization.

[5] Xu, Kun, Ming Yan Jiang, and Dong Feng Yuan. "Parallel artificial bee colony algorithm for the traveling salesman problem." Advanced Materials Research 756 (2013): 3254-3259.

[6] Hoffman, Karla L., Manfred Padberg, and Giovanni Rinaldi. "Traveling salesman problem." Encyclopedia of operations research and management science 1 (2013): 1573-1578.

[7] Pardalos, P.M., Du, D. and Graham, R.L. (2013) Handbook of Combinatorial Optimization. New York: Springer.

[8] Larson, Richard C., and Amedeo R. Odoni. Urban operations research. No. Monograph. 1981.

[9] Papadimitriou, Christos H. "The Euclidean travelling salesman problem is NP-complete." Theoretical computer science 4.3 (1977): 237-244.

[10] Mohd Razali, Noraini. Genetic algorithm for process sequencing modelled as the travelling salesman problem with precedence constraints. Diss. Dublin City University, 2014.

[11] J. K. Lenstra & A. H. G. Rinnooy Kan (1975) Some Simple Applications of the Travelling Salesman Problem, Journal of the Operational Research Society, 26:4, 717-733, DOI: 10.1057/jors.1975.151.

[12] Burkard, Rainer E., et al. The quadratic assignment problem. Springer US, 1998.

[13] Loiola, Eliane Maria, et al. "A survey for the quadratic assignment problem." European journal of operational research 176.2 (2007): 657-690.

[14] Šimko, Dávid. "METHODS OF DISTRIBUTION, LAYOUT AND HUNGARIAN METHOD." Acta logistica 3.1 (2016): 9-13.

[15] A. Shukla, "A modified bat algorithm for the Quadratic Assignment Problem," 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 2015, pp. 486-490, doi: 10.1109/CEC.2015.7256929.

[16] Zlatev, Zahari, et al., eds. Large-scale computations in air pollution modelling. Vol. 57. Springer Science & Business Media, 1999.

[17] Pitsoulis, Leonidas Sofoklis. Algorithms for nonlinear assignment problems. University of Florida, 1998.

[18] Delahaye, Daniel, Supatcha Chaimatanan, and Marcel Mongeau. "Simulated annealing: From basics to applications." Handbook of metaheuristics (2019): 1-35.

[19] Prajapati, Vishnu Kumar, Mayank Jain, and Lokesh Chouhan. "Tabu search algorithm (TSA): A comprehensive survey." 2020 3rd International Conference on Emerging Technologies in Computer Engineering: Machine Learning and Internet of Things (ICETCE). IEEE, 2020.

[20] Gendreau, Michel, and Jean-Yves Potvin. "Tabu search." Search methodologies: introductory tutorials in optimization and decision support techniques (2005): 165-186.

[21] M. Gendreau, A. Hertz and G. Laporte, New insertion and post optimization procedures for the traveling salesman problem, CRT-708 Centre de Recherche sur les Transports, Universite de Montreal, Montreal, Canada (1991).

[22] B.K.A. Ngoi, M.L. Tay, E.S. Chua, Applying spatial representation techniques to the container packing problem, International Journal of Production Research 32 (1994) 111–123.

[23] D. Pisinger, Heuristics for the container loading problem, European Journal of Operational Research 141 (2002) 143–153.

[24] Y. Rochat, E. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1 (1995) 147– 167.

[25] Brandão, José. "A tabu search algorithm for the open vehicle routing problem." European Journal of Operational Research 157.3 (2004): 552-564.

[26] Brandao, José. "A new tabu search algorithm for the vehicle routing problem with backhauls." European Journal of Operational Research 173.2 (2006): 540-555.

[27] Marín, A. Discrete location for bundled demand points. TOP 18, 242–256 (2010). https://doi.org/10.1007/s11750-009-0097-0

[28] Gandomi, Amir Hossein, and Amir Hossein Alavi. "Krill herd: a new bio-inspired optimization algorithm." Communications in nonlinear science and numerical simulation 17.12 (2012): 4831-4845.

[29] Hahsler, Michael, Bettina Grün, and Kurt Hornik. "arules-A computational environment for mining association rules and frequent item sets." Journal of statistical software 14.15 (2005): 1-25.

[30] Laszlo Szathmary. Symbolic Data Mining Methods with the Coron Platform. Software Engineering [cs.SE]. Université Henri Poincaré - Nancy 1, 2006. English. NNT : 2006NAN10159 . tel-01754284v2 [43] Pandey, Shreelekha, and Pritee Khanna. "Content-based image retrieval embedded with agglomerative clustering built on information loss." Computers & Electrical Engineering 54 (2016): 506-521.

[31] Nejad, Anahita Sabagh, and Gabor Fazekas. "Solving a traveling salesman problem using meta-heuristics." IAES International Journal of Artificial Intelligence (IJ-AI) 11.1 (2022): 41.

[32] Rokach, Lior. "A survey of clustering algorithms." Data mining and knowledge discovery handbook (2010): 269-298.

[33] Jin, X., Han, J. (2016). K-Medoids Clustering. In: Sammut, C., Webb, G. (eds) Encyclopedia of Machine Learning and Data Mining. Springer, Boston, MA. https://doi.org/10.1007/978-1-4899-7502-7_432-1

[34] G. Ramadevi, S. Yeruva, P. Sravanthi, P. E. Vamsi, and S. J. Prakash, “Analysis and detection of diabetes using datamining techniques – efficiency comparison,” International Journal of Scientific Research in Science and Technology,vol. 7, no. 4, pp. 73–79, Jul. 2021, doi: 10.32628/cseit217425

[35] X. Xu, M. Ester, H. P. Kriegel, and J. Sander, “Clustering and knowledge discovery in spatial databases,” Vistas in Astronomy, vol. 41, no. 3, pp. 397–403, Jan. 1997, doi: 10.1016/S0083-6656(97)00044-5

[36] P. K. Prasad and C. P. Rangan, “Privacy-preserving BIRCH algorithm for clustering over vertically partitioned databases,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and lecture Notes in Bioinformatics), vol. 4165 LNCS, Springer Berlin Heidelberg, 2006, pp. 84–99.

[37] H. Okumura and K. Naito, “Weighted kernel estimators in nonparametric binomial regression,” Journal of Nonparametric Statistics, vol. 16, no. 1–2, pp. 39–62, Feb. 2004, doi: 10.1080/10485250310001624828

[38] L. Cao, J. Z. Huang, J. Bailey, Y. S. Koh, and J. Luo, New frontiers in applied data mining: PAKDD 2011 International-Workshops, Shenzhen, China, May 24-27, 2011, Revised Selected Papers, vol. 7104, no. September 2014. SpringerBerlin Heidelberg, 2012.

[39] Gandomi, Amir Hossein, et al., eds. Metaheuristic applications in structures and infrastructures. Newnes, 2013.

[40] Yang, Xin-She. Nature-inspired optimization algorithms. Academic Press, 2020.

[41] Goldberg, David E. "Genetic algorithms in search, optimization, and machine learning." (1989)

[42] Darwin, Charles. "On the origins of species by means of natural selection." London: Murray (1859).

[43] Pandey, Hari Mohan (11.2014). "A comparative review of approaches to prevent premature convergence in GA". Applied soft computing(1568-4946), 24, p. 1047.

[44] Razali, Noraini Mohd, and John Geraghty. "Genetic algorithm performance with different selection strategies in solving TSP." Proceedings of the world congress on engineering. Vol. 2. No. 1. Hong Kong, China: International Association of Engineers, 2011.

[45] Holland, John H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.

[46] Shukla, Anupriya, Hari Mohan Pandey, and Deepti Mehrotra. "Comparative review of selection techniques in genetic algorithm." 2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE). IEEE, 2015.

[47] Sivaraj, R., and T. Ravichandran. "A review of selection methods in genetic algorithm." International journal of engineering science and technology 3.5 (2011): 3792-3797.

[48] Yu, Fengrui, et al. "Improved fitness proportionate selection-based genetic algorithm." 2016 3rd International Conference on Mechatronics and Information Technology. Atlantis Press, 2016.

[49] Larranaga, Pedro, et al. "Genetic algorithms for the traveling salesman problem: A review of representations and operators." Artificial intelligence review 13 (1999): 129-170.

[50] K. De Jong, W. M. Spears, “Using Genetic Algorithms to Solve NP-Complete Problems,” Proceedings of the Third International Conference on Genetic Algorithm, Morgan Kaufman, Los Altos, CA, 124 – 132, 1989

[51] Pandey, Hari Mohan. "Performance evaluation of selection methods of genetic algorithm and network security concerns." Procedia Computer Science 78 (2016): 13-18.

[52] Yu, Fengrui, et al. "Improved roulette wheel selection-based genetic algorithm for TSP." 2016 International conference on network and information systems for computers (ICNISC). IEEE, 2016.

[53] Applegate, David, et al. "Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems." Mathematical programming 97 (2003): 91-153.

[54] Shimodaira, Hisashi. "A new genetic algorithm using large mutation rates and population-elitist selection (GALME)." Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence. IEEE, 1996.

[55] H. Chiroma et al., "Bio-inspired computation: Recent development on the modifications of the cuckoo search algorithm," Applied Soft Computing, vol. 61, pp. 149–173, Dec. 2017, doi: 10.1016/j.asoc.2017.07.053. [22]

[56] X.-S. Yang and S. Deb, "Cuckoo search: recent advances and applications," Neural Computing and Applications, vol. 24, no. 1, pp. 169–174, Jan. 2014, doi: 10.1007/s00521-013-1367-1.

[57] S. Ghafori and F. S. Gharehchopogh, "Advances in spotted hyena optimizer: a comprehensive survey," Archives of Computational Methods in Engineering, Jul. 2021, doi: 10.1007/s11831-021-09624-4.

[58] Wang, Dongshu, Dapei Tan, and Lei Liu. "Particle swarm optimization algorithm: an overview." Soft computing 22 (2018): 387-408.

[59] Kalam, Akhtar, et al. "Intelligent Computing Techniques for Smart Energy Systems Proceedings of ICTSES 2018." Proceedings of ICTSES (2018): 1.

[60] Sabagh Nejad, A.: Data Clustering Using Hybrid Algorithm. In: The 11th International Conference on Applied Informatics (ICAI 2020). Ed.: Fazekas Istvan, Kovasznai Gergely, Eszterhazy Karoly Egyetem, Eger, 1-8, 2020.

[61] Khosravanian, Rassoul, et al. "A comparative study of several metaheuristic algorithms for optimizing complex 3-D well-path designs." Journal of Petroleum Exploration and Production Technology 8 (2018): 1487-1503.

[62] Qiongshuai, Lv, and Wang Shiqing. "A hybrid model of neural network and classification in wine." 2011 3rd International Conference on Computer Research and Development. Vol. 3. IEEE, 2011.

[63] Vardhan, Mr M. Vishnu, and P. Sangameswararaju. "ABC and NFC based unified power quality conditioner for compensating power quality problem."

[64] Jaradat, Ameera, and Waed Diabat. "Solving traveling salesman problem using firefly algorithm and k-means clustering." 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT). IEEE, 2019.

[65] Panigrahi, Bijaya Ketan, et al., eds. Swarm, Evolutionary, and Memetic Computing: First International Conference on Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, Chennai, India, December 16-18, 2010, Proceedings. Vol. 6466. Springer, 2010.

[66] Johari, Nur Farahlina, et al. "Machining parameters optimization using hybrid firefly algorithm and particle swarm optimization." Journal of Physics: Conference Series. Vol. 892. No. 1. IOP Publishing, 2017.

[67] Kumar, Vijay, and Dinesh Kumar. "A systematic review on firefly algorithm: past, present, and future." Archives of Computational Methods in Engineering 28 (2021): 3269-3291.

[68] Hofmann, Eileen E., et al. "Lagrangian modelling studies of Antarctic krill (Euphausia superba) swarm formation." ICES Journal of Marine Science 61.4 (2004): 617-631.

[69] Wang, Gaige, et al. "Incorporating mutation scheme into krill herd algorithm for global numerical optimization." Neural Computing and Applications 24 (2014): 853-871.

[70] Li, Qin, and Bo Liu. "Clustering using an improved krill herd algorithm." Algorithms 10.2 (2017): 56.

[71] Cheng, Long, Xue-han Wu, and Yan Wang. "Artificial flora (AF) optimization algorithm." Applied Sciences 8.3 (2018): 329.

[72] Sayed, Gehad Ismail, Ashraf Darwish, and Aboul Ella Hassanien. "A new chaotic whale optimization algorithm for features selection." Journal of classification 35 (2018): 300-344.

[73] Angelov, Plamen Parvanov, ed. Handbook On Computer Learning And Intelligence (In 2 Volumes). World Scientific, 2022.

[74] Singh, Puja, and Shashi Prakash. "Optical network unit placement in Fiber-Wireless (FiWi) access network by Whale Optimization Algorithm." Optical Fiber Technology 52 (2019): 101965.

[75] N. Rana, M. S. A. Latiff, S. M. Abdulhamid, and H. Chiroma, "Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments," Neural Computing and Applications, vol. 32, no. 20, pp. 16245–16277, Oct. 2020, doi: 10.1007/s00521-020-04849-z.

[76] N. Al-Madi, H. Faris, and S. Mirjalili, "Binary multi-verse optimization algorithm for global optimization and discrete problems," International Journal of Machine Learning and Cybernetics, vol. 10, no. 12, pp. 3445–3465, Dec. 2019, doi: 10.1007/s13042- 019-00931-8.

[77] Z. Zhang, C. Huang, K. Dong, and H. Huang, "Birds foraging search: a novel population-based algorithm for global optimization," Memetic Computing, vol. 11, no. 3, pp. 221–250, Sep. 2019, doi: 10.1007/s12293-019-00286-1.

[78] Prasad, Dharmbir, Aparajita Mukherjee, and V. Mukherjee. "Transient stability constrained optimal power flow using chaotic whale optimization algorithm." Handbook of neural computation. Academic Press, 2017. 311-332.

[79] Shi, Y., Tan, Y. and Tang, Q. (2018) Advances in swarm intelligence 9th international conference, ICSI 2018, Shanghai, China, June 17-22, 2018, Proceedings, part II. Cham: Springer International Publishing.

[80] Song, H., and Jiang, D. (eds.) (2019) '11th International Conference, SIMUtools 2019, Chengdu, China, July 8–10', in Simulation tools and Techniques. Springer International Publishing.

[81] Sabagh Nejad, Anahita, and Gábor Fazekas. "Reducing the time needed to solve a traveling salesman problem by clustering with a Hierarchy-based algorithm."

[82] S. George, "Mp-testdata—the tsplib symmetric traveling salesman problem instances." 2008.

[83] Kovács, László, Anita Agárdi, and Bálint Debreceni. "Efficiency Analysis of the Vertex Clustering in Solving the Traveling Salesman Problem." Annales Mathematicae et Informaticae. Vol. 48. No. 1. 2018.