People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arreridj
Faculty of Mathematics and Informatics
Informatics Department

DISSERTATION

Presented in fulfillment of the requirements of obtaining the degree

Master in Informatics

Specialty: Networks and Multi Media

THEME

A Hybrid Ant Colony Optimization for Multidepot Vehicle Routing Problem

Presented by :

Abderrahim BOUTALBI

Saber BENNIA

Publicly defended on:

In front of the jury composed of:

President: NOUIOUA Farid

Examiner: BOUTOUHAMI Sara

Supervisor: SAHA Adel

2021/2022

Dedications I

This thesis is dedicated to

The sake of Allah, my Creator and my Master,

My great teacher and messenger, Mohammed (May Allah bless and grant him), who taught us the purpose of life,

The University Mohamed El Bachir El Ibrahimi; my second magnificent home;

My great parents, who never stop giving of themselves in countless ways,

My dearest grand mather, who leads me through the valley of darkness with the light of hope and support,

My beloved sisters; particularly my dearest brother, Mouhamed, who stands by me when things look bleak,

My beloved friends: Islem , and Sofaine, whom I can't force myself to stop loving. To all my family, the symbol of love and giving,

My friends who encourage and support me,

All the people in my life who touch my heart, I dedicate this research

Dedications II

I dedicate this modest work in principle to the closest people

of my life ; my father and my mother who were present for me

throughout my education and my life, but above all who have

always knew how to find the words that encouraged me

and who pushed me to go forward

The University Mohamed El Bachir El Ibrahimi; my second magnificent home;

I also dedicate this work to all my family who were always behind me.

to strengthen me during my difficult moments, in particular:

My brothers: AHMED, KHALED

I also want to dedicate this work to my dear friends who have never forgotten us and who were there when needed: hocine, otmane, sofiane

Finally to all the people who are dear to me and whom I love.

saber

Acknowledgment I

All praise and thanks are only for Allah, the One who, by his blessing and favor, perfected goodness good works are accomplished

In completing this project successfully, a lot of people helped me. I would like to thank all those involved in this project.

First of all, I would like to thank God for his ability to successfully complete this project. Then I will thank my supervisor SAHA Adel for his efforts during this work, I would like to thank my friend Eng. Sofian METTAI who taught me a lot about this project by guiding me. His suggestions and guidance helped accomplish this course.

Acknowledgment II

First of all, I would like to thank God.

I thank the Almighty and the Merciful, who gave me life and strength, as well as the courage and audacity to overcome all the difficulties, to have followed and accomplished all the stages of my studies.

Secondly, I would like to thank my supervisor, Saha Adel, for having assisted me and helping me with their precious advice and for having offered me all the possibilities, such as following and providing the necessary documentation for the realization of my thesis study.

Finally, I would also like to thank all my friends and

My colleagues who assist me directly or indirectly in realizing

This work

Abstract

In this thesis, we proposed a hybrid Ant Colony Optimization for Multi Depot Vehicle Routing Problem using the Nearest Distance Cluster Algorithm, 2-opt and mutation operation in order to Gain the distance and thus reduce the cost in general with the condition of the limited capacity of vehicles

Key words: HACO, ACO, VRP, hybrid Algorithm.

Résumé

Dans ce mémoire, nous avons proposé une optimisation hybride de colonies de fourmis pour le problème de routage dynamique de véhicules multi dépôts à l'aide de l'algorithme de cluster de distance la plus proche, Opération d'échange local (2-opt) et opération de mutation afin de Gagner de la distance et ainsi réduire le coût en général avec la condition de la capacité limitée des véhicules

Mots clés : OCFH, OCF, PTV, Algorithme hybride.

ملخص

في هذه المذكرة، اقترحنا تحسينًا هجينًا لمستعمرة النمل لمشكلة توجيه المركبات الديناميكية متعددة النقاط باستخدام خوارزمية أقرب مسافة للمجموعة ، عملية التبادل المحلي (2-opt) وعملية الطفرة من أجل كسب المسافة وبالتالي تقليل التكلفة بشكل عام مع اخذ بعين الاعتبار السعة المحدودة للمركبات

الكلمات المفتاحية: تحسين مستعمرة النمل الهجين ، تحسين مستعمرة النمل ، مشكلة توجيه المركبة ، الخوارزمية الهجينة

Abreviations list

NP: Nondeterministic polynomial (not deterministic polynomial time)

MILP: Linear program in mixed variables (Mixed Integer Linear Problem)

PLI: Linear program in continuous variables (Integer Linear Program)

0- 1ILP: Linear program in binary variables (0-1 Integer Linear Program)

Cop: Combinatorial optimization problems

ACO	Ant Colony Optimization (Algorithm)
TSP	Travelling Salesmen Problem
H-ACO	Hybrid-Ant Colony Optimization (Algorithm)
CMDVRP	Capacitated Multi Depot Vehicle Routing Problem considering
CVRP	Capacitated Vehicle Routing Problem
VRPTW	Vehicle Routing Problem Time Windowed
VRPPD	Vehicle Routing Problem with pick-up and delivery
GSP	Group-shop Scheduling Problem
PFSP	Permutation Flow Shop Problem
FAP	Frequency Assignment Problem
MDVRP	Multi Depot Vehicle Routing Problem
PSO	Particle Swarm Optimization (Algorithm)
GA	Genitic Algorithm
2-opt	Two arc optimizations (Algorithm)

VRP : Vehicle Routing Problem

List of contents

List of tables

List of figures

Dissertation

Due to the rapid development of technology and in the purpose to improving service and bringing it closer to customers, many delivery companies use multi-depot, which leads to overall cost reduction (because now the distance between depots and the customer is small + delivery speed is better)

Meanwhile, due to the actual constraints of service hours and service distances, and in the purpose to improve the delivery service, companies usually build multiple depots to serve a large number of customers.

Hybrid ant colony optimization (HACO) is improved by the nearest distance clustering approach and local exchange, a nearest distance-based clustering approach is an optimization technique proposed to allocate all customers to their nearest depot, and the local interchange operation to minimize the distance to travel in the Vehicle Routing Problem (VRP).

Finally, a test is applied to evaluate the proposed algorithm; in the meantime, and the total cost of the solution.

General Introduction

Recently, due to the widespread of delivery services in various commercial activities in our country, as well as transportation services in various economic companies, due to their great importance in increasing production efficiency and increasing habits, and in particular improving the quality of services

So many production companies and delivery service companies impose many problems with delivery and vehicle routing and other likely problems Often this involves finding the maximum or minimum value of some function: the minimum time to make a certain journey, the minimum cost for doing a task, and the maximum power that can be generated by a device.

Many of these problems can be solved by finding the appropriate function and then using techniques of calculus to find the maximum or the minimum value required In order to develop the company's return as well as reduce spending on customers, and by mentioning the problems of improvement(combinatorial optimization problems), we mention the most famous of these problems: Cutting Stock problem, Packing Problems, Minimum spanning tree, vehicle routing problem, travelling salesman problem, back wallet problem …..and so on.

In this graduation note we will mention some of these problems, their description and formulation also some model of these problems, some methods of resolving this kind of problems and also, we mention ACO ant colony optimization the algorithm used to resolve this kind of problems and address in particular the ACO for solving CMDVRP (ant colony optimization algorithm for solving multi depot vehicle routing problem).

Chapter 01: Combinatorial optimization

1. 1. Introduction

Combinatorial optimization occupies a very important place in operations research, in discrete mathematics and in computer science. Its importance is justified on the one hand by the great difficulty of the optimization problems and on the other hand by the many practical applications that can be formulated in the form of an optimization problem. Although combinatorial optimization problems are often easy to define, they are usually difficult to solve. Indeed, most of these problems belong to the class of NP-hard problems and therefore do not currently have an effective algorithmic solution valid for all data. [2]

Given the importance of these problems, many resolution methods have been developed in operations research (OR) and artificial intelligence (AI). These methods can be broadly classified into two main categories: exact methods (complete) which guarantee the completeness of the resolution and approximate methods (incomplete) which lose completeness to gain in efficiency, so in this chapter, we talk more detail about the techniques and optimization methods used to solve our problem

1.2. Optimization:

1.2.1. Definition of optimization

The art of understanding a real problem, of being able to transform it into a mathematical model that can be studied in order to extract its structural properties and characterize the solutions to the problem, it is the art of exploiting this characterization in order to determine the algorithms which calculate them but also to highlight the limits on the efficiency and the effectiveness of these algorithm [3].

1.2.2. Combinatorial optimization

Combinatorial optimization is a mathematical technique, which consists in minimizing or maximizing an objective function (Cost, time, distance, etc.). Whose goal is to find and define an optimal solution most suitable for optimization from a set of possible solutions [3].

1.2.3. Examples of optimization problems:

1.2.3.1. Backpack problem

The "Backpack problem" is a selection problem that consists in maximizing a quality criterion under a linear resource capacity constraint. It owes its name to the analogy that can be made with the problem that arises for the hiker when filling his backpack: he must choose the objects to carry so as to have the most "useful" bag. possible, while respecting its volume.

More formally, it can be described as follows. Given a set of n elements and a resource available in limited quantity, b. For j = 1 to n, we denote pj the profit associated with the selection of element j and we denote aj the quantity of resource that element j requires, if it is selected. The coefficients pj and aj take positive values for all j = 1 to n. The knapsack problem consists in choosing a subset of the n elements, which maximizes the total profit obtained, respecting the quantity of available resource.

Each element j is associated with a selection variable, x_j, binary, equal to 1 if j is selected, equal to 0 otherwise. the total profit obtained can then be written as the sum:

$\sum_{j=1}^{n} p_j \ x_j$ and the total amount of resource used as the sum: $\sum_{j=1}^{n} a_j \ x_j$The backpack problem is therefore modeled as:

$$MAX \sum_{j=1}^{n} p_j \ x_j \qquad 1.(1)$$

$$\sum_{j=1}^{n} a_j \ x_j \leq b \qquad 1.(2)$$

$$x_j \in \{0,1\} \ \forall j = 1..n \qquad 1.(3)$$

The resource constraint is called the "backpack constraint"; it is found in optimization problems, from many fields of application, which involve resources with limited capacity.

In the case where there are several constraints of this type (for example, the hiker can consider not only a maximum volume, but also a maximum weight, that his bag can support), we speak of a backpack problem. "multidimensional".

The knapsack problem has been the subject of various works proposing exact methods of resolution. The proposed algorithms fall under three main types of methods. First, separation and evaluation 1 type algorithms were proposed in the 1970s, allowing to deal efficiently with small instances. These performances were subsequently improved by adding additional constraints to reinforce the bounds in the search tree. Secondly, algorithms based on the identification of a critical variable and an associated subset of variables, on which a truncated tree search is applied, have made it possible, from the 1980s, to increase the size of the instances that can be resolved (up to n=100000). Third, efficient dynamic programming algorithms, dynamic programming is combined with the identification of a critical variable and the use of bound strengthening techniques. [5]

1.2.3.2. Assignment problem

The "assignment problem" consists in establishing links between the elements of two distinct sets, in such a way as to minimize a cost and while respecting link uniqueness constraints for each element.

We consider m tasks and n agents, with n ≥ m. for any pair (i ,j) (i = 1 to m, j = 1 to n), the assignment of task i to j entails a performance cost noted $c_{i.j}$($c_{i.j}$≥ 0). each task must be performed exactly once and each agent can perform at most one task. the problem consists in assigning the tasks to the agents, to minimize the total cost of realization and respecting the constraints of realization of the tasks and availability of the agents.

To any task/agent pair (i , j), we associate an assignment variable, $x_{i,j}$, binary, which takes the value 1 if task i is assigned to agent j and 0 otherwise. The total cost of carrying out the tasks is then expressed by the sum: $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i.j}\, x_{i,j}$. The number of agents performing task

i is given by: $\sum_{j=1}^{n} x_{i,j}$ for all i = 1 to m and the number of tasks performed by agent j is given by: $\sum_{i=1}^{m} x_{i,j}$, for all j = 1 to n. We can therefore model the assignment problem in the form:

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i.j}\, x_{i,j} \qquad 1.(4)$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \ (\forall \text{ i } = \ 1 \,..\, \text{m}) \qquad 1.(5)$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \quad (\forall \text{j } = \ 1..\, \text{n}) \qquad 1.(6)$$

$$x_{i,j} \in \{0,1\} \, \forall i \ = \ 1..m \, , \forall j \ = \ 1..n \qquad 1.(7)$$

The constraints of this problem are found in many applications involving resource allocation problems. They are generally called "assignment constraints".

In graph theory, we can reduce ourselves to a coupling problem in a bipartite graph. A graph G is said to be bipartite if one can divide the vertices into two sets X1 and X2 such that all edges in the graph join a vertex of X1 to a vertex of X2. A "matching" in a bipartite graph is a set of edges which have, 2 by 2, no common vertex in G.

By associating X1 to the set of tasks, of cardinality m and X2 to the set of agents, of cardinality n, an edge (i ,j) in the graph G (with i ∈ X1 and j ∈ X2) represents the possibility of assigning task i to agent j; the weight is associated $c_{i.j}$with each edge (i,j) of G. The weight of a matching being defined as the sum of the weights of its edges, the assignment problem then amounts to finding a matching of cardinality m with minimal weight in the graph G.

The particular case where X1 and X2 have the same cardinality (corresponding to the case n = m for the assignment problem) is frequently studied; we are then interested in the search for a coupling of maximum cardinality. If we consider sets X1 and X2 of cardinality n and if there are n 2 edges in the graph G (the bipartite graph is complete), then the matching maximum is of cardinality n and it is called "perfect matching". We can extend this problem to that of the search for a maximal matching of minimal weight in G.

The assignment, or coupling, problem in a bipartite graph can be modeled as a minimum-cost maximum flow problem in which the capacities of the arcs are all equal to 1.

This is a classic problem in graph theory that amounts to seeking to pass a maximum throughput through a network, for a lower cost 2 . Simple and effective algorithms exist to solve this problem; in particular the algorithm of Busaker and Gowen which starts from a null flow and which increases it progressively by searching for " increasing chains " (ie, paths on the arcs of which one can systematically increase the flow), of minimal cost.

The "Hungarian method", proposed by Kuhn in 1955, is a dual algorithm which is based on modeling the assignment problem in the form of a linear program, but which can be seen as a variant of Busaker 's algorithm and Gowen , specialized for the bipartite structure of the graph. Because of its great efficiency on this type of problem, it is the reference algorithm in Operations Research to solve the assignment problem. Its principle is based on the fact that the pairings of minimal weight in the graph of the primal problem are exactly the pairings of maximum cardinality in the graph of the dual. [5]

1.2.3.3 Traveling salesman problem

The "traveling salesman problem", or TSP (for Traveling Salesman Problem), is the following: a sales representative having (n) cities to visit wishes to establish a route which allows him to pass exactly once by each city and to return to his starting point for a lower cost, that is- that is, by covering the shortest possible distance. It is one of the oldest and most widely studied problems in combinatorial optimization. Its applications are numerous. For example, problems of manufacturing process sequencing or path optimization in robotics can be expressed directly in the form of a TSP and certain problems, such as transport problems, are more complex than the TSP but present a structure underlying type TSP.

Let G = (X ,U), a graph in which the set X of vertices represents the cities to be visited, as well as the starting city of the tour, and U, the set of arcs of G, represents the possible routes between cities. To any arc (i ,j) ∈ U, we associate the travel distance from $d_{i.j}$city i to city j. The length of a path in G is the sum of the distances associated with the arcs of this path. The TSP is then reduced to finding a Hamiltonian circuit (ie, a closed path passing exactly once through each of the vertices of the graph) of minimal length in G. In the case where there exist certain arcs (i ,j) ∈ U for which $d_{i.j} \neq d_{j,i}$, we speak of asymmetric TSP.

We can formulate the TSP in an equivalent way by associating to each pair (i,j) of cities to visit (i = 1 to n, j = 1 to n and i ≠j) a distance $\delta_{i.j}$equal to $d_{i.j}$if there is a means of go directly from i to j (ie, (i,j) ∈ U in G) , fixed at ∞ otherwise and a succession variable, $x_{i,j}$, binary, which takes the value 1 if the city j is visited immediately after the city i in the tour and which takes the value 0 otherwise. The TSP is then modeled by:

$$Min \sum_{i=1}^{m} \sum_{j=1}^{n} \delta_{i.j}\, x_{i,j} \qquad 1.(8)$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \;\forall \text{ i } = \text{ 1..n} \qquad 1.(9)$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \quad \forall \text{j } = \text{ 1..n} \qquad 1.(10)$$

$$\sum_{i \epsilon s, j \notin s} x_{i,j} \geq 2 \qquad \forall s \subset X, S \neq \emptyset \quad 1.(11)$$

$$x_{i,j} \in [0,1] \;\forall \text{ i} = 1..\text{n} \text{ , } \forall \text{j} = 1..\text{n}$$

The first two constraints reflect the fact that each city must be visited exactly once; the third constraint prohibits solutions composed of disjoint subtours, it is generally called constraint of elimination of subtours.

Integer Linear Programming algorithms have been developed to solve the TSP exactly

In particular, separation and evaluation search methods are effectively reinforced by the addition of cuts (branch and cut algorithms) and constraint propagation techniques in the search tree. Dynamic programming algorithms for finding Hamiltonian circuits in a graph, as well as Constraint Programming (CPP) also provide good results for problems of up to a hundred nodes. In addition, efficient heuristic procedures and local optimization techniques are available (e.g. Lin and Kernighan 's algorithm , as well as lower bound calculations, based for example on a Lagrangian relaxation of the TSP, making it possible to provide a good framework of the optimum.

There are many variants of the TSP, obtained either by adding constraints – such as the TSP with time windows (TSPTW for Traveling Salesman Problem with Time Windows) in which each city must be visited within a given time interval – either by modification, such as

vehicle routing problems (VRP for Vehicle Routing Problem) in which we no longer consider a single sales representative to visit the cities but a team (a fleet of vehicles) and which can be seen as flow problems. [5]

1.2.3.4. Scheduling problem

The "scheduling problem" consists in sequencing and placing in time a set of activities (elementary work entities), taking into account temporal constraints (deadlines, sequence constraints, etc.) and constraints relating to the use and availability of the resources required by the activities .Posed in this way, it is a problem of satisfaction of constraints which finds its applications in various fields (project management, production workshops, etc.) and which is the subject of research work from a point of view. view of decision support, in particular through constraint-based approaches .In an optimization context, we also seek to minimize (or maximize) a criterion, such as for example the total duration of the activities (minimization of Makespan).

The term "scheduling problem", unlike the three problems seen previously, does not refer to a totally defined problem for which there is a direct mathematical formulation, but rather to a family of problems. Indeed , a scheduling problem is defined by the data of the activities and the resources which constitute it and these elements can be of very varied natures.

For example, a resource is "disjunctive" (or non-sharable) if it cannot perform more than one activity at a time (this is called a disjunctive scheduling problem), otherwise, the resource is said to be "cumulative" (and leads to a cumulative scheduling problem). A resource is "renewable" if, after having been used by one or more activities, it is again available in the same quantity (machine, processor ...); the amount of resource usable at any time is limited. If the performance of an activity reduces it by a certain quantity, the resource is on the contrary "consumable" (raw materials, budget, etc.) and in this case, the overall consumption over time is limited. A resource is "doubly constrained" if both its instantaneous use and its overall consumption are limited (project funding is the most typical example).

Regarding the activities, we can consider that it is possible to interrupt them and then resume them later, or on the contrary that they must be carried out without interruption. One then speaks respectively of a "preemptive" and "non-preemptive" problem. different constraints may apply to activities; for example, execution "window" constraints, with "earliest start" and " latest end" dates, "precedence constraints" which impose that certain activities be fi nished before others can begin, or time constraints between two activities (limited waiting time or, on the contrary, preparation time required between two tasks).

Several types of decision variables can be used to model a scheduling problem. For example, in the case of a one-machine, disjunctive and non-preemptive scheduling problem, presents a review of the different mathematical formulations proposed in the literature. There are four types of decision variables: real variables indicating the start or end of the execution of an activity on the machine (ti is the start date of execution of activity i), binary variables indicating an immediate succession relationship between two tasks ($f_{i,j} = 1$if activity j is executed just after i on the machine,$f_{i,j} = 0$ otherwise), binary variables indicating the position of an activity in the sequence ($s_{i,j} = 1$if activity i is the $j^{\text{ème}}$activity executed by the machine, $s_{i,j} = 0$otherwise), binary variables, indexed on the discretized time and indicating the start of an activity at a certain time step ($x_{i,t} = 1$if activity i starts at the time step $x_{i,t} = 0$if otherwise).

The various constraints are expressed more or less easily in the various formalisms. For example, a formulation based solely on the start dates of activities makes it possible to formulate precedence constraints of the type "activity i is carried out before activity j" ($t_j - t_i > p_i$, where $p_i$is the duration of execution of activity i), but does not make it possible to express certain relationships, such as the preparation times between two activities. In this case, it is necessary to introduce a variable indicating the succession of activities between them (variable of the "successor" or "position in the sequence" type).

We thus obtain linear models for a large number of scheduling problems. The linear relaxation of these models provides bounds which can be used directly inside a tree search algorithm by separation and evaluation, or which can be refined by different procedures (Lagrangian relaxation, addition of cuts). In general, purely linear techniques do not give good

results on scheduling problems. The most efficient algorithms , whether for an exact resolution of the optimization problem or for the search for admissible solutions of good quality, are most often based on the use of advanced techniques of constraint propagation.[5]

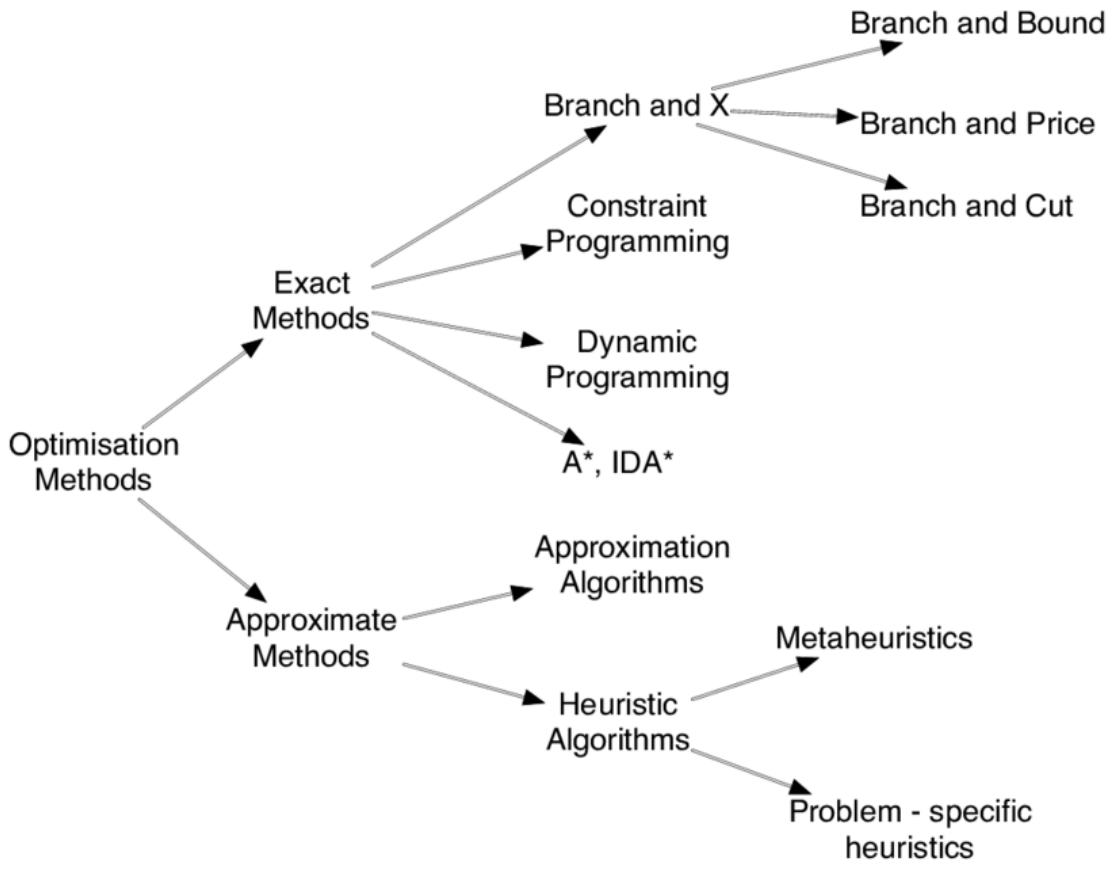

figure 1. 1Methods for solving combinatorial optimization problems.[2]

1.3. Combinatorial optimization problems resolution methods

1.3.1. Exact methods

Exact resolution methods are numerous and are characterized by the fact that they make it possible to obtain one or more solutions whose optimality is guaranteed. Among these methods, we can notice the simplex algorithm which makes it possible to

obtain the optimal solution of a problem by going through the convex closure of the search set (set of admissible solutions) and this by passing from vertex to vertex. Despite having a non-polynomial worst-case mathematical complexity, it solves most problems quickly. However, it can only be applied to problems having the property of convexity, i.e. to problems in continuous variables or to problems in integer variables having a unimodular constraint matrix T (because in this case, all the vertices of the search set are integers) such as transport or assignment problems.

For other problems (ILP, MILP, 0-1ILP), there are several methods:

1.3.1.1. Dynamic programming

- Dynamic programming consisting in placing the problem in a family of problems of the same nature but of different difficulty then in finding a recurrence relation linking the optimal solutions of these problems.

1.3.1.2. Branch & Bound

- Branch & Bound consisting of making an implicit enumeration by separating the problem into sub-problems and evaluating these using relaxation (mainly continuous or Lagrangian) until you only have problems that are easy to solve or which we know with certainty that they cannot contain an optimal solution.

1.3.1.3. Polyhedral methods

- Polyhedral methods consisting of gradually adding additional constraints in order to reduce the domain of admissible solutions to a convex domain (without removing the optimal solution(s) of course).

These methods are general and often require particularization vis-à-vis a specific problem. There are also generic applications (AMPL, CPLEX, LINDO, MPL, OMP, XPRESS...) allowing solving all the problems that can be written in the algebraic form of a problem in binary, integer

Or mixed variables.

It should also be noted that the method consisting in carrying out an explicit enumeration of all the solutions (i.e. testing them one by one, a method that can be envisaged for all the problems with variables with bounded values) very quickly shows its limits as soon as that the number of variables increases since its complexity is in kn where k represents the number of values that a variable can take and n the number of variables of the problem [4].

1.3.2. Approximate methods

In certain situations, it is necessary to have a good quality solution (that is to say quite close to the optimum) in a context of limited resources (computing time and/or memory). In this case the optimality of the solution will not be guaranteed, nor even the difference with the optimal value. However, the time necessary to obtain this solution will be much lower and could even be fixed (obviously in this case the quality of the solution obtained will strongly depend on the time left to the algorithm to obtain it).

Typically, this type of method, called heuristic is particularly useful for problems requiring a real-time (or very short) solution or for solving difficult problems on large numerical instances. They can also be used to initialize an exact method (Branch & Bound for example).

Among these methods, it is necessary to distinguish the heuristics targeted on a particular problem and the more powerful and adaptable metaheuristics to solve a large number of problems. However, a metaheuristic, to be sufficiently efficient on a given problem, will require a more or less fine adaptation. These approximate methods can be classified into different categories:

- Constructive (greedy algorithms, Pilot method, GRASP)

Local search (descent algorithms, multi-starts, simulated annealing, threshold algorithm, Tabu search, sound effect method)

- Evolutionists (Genetic Algorithms, Evolution Algorithms, Scattered Search, Path Method, Ant Systems)

- Neural networks (Hopfield - Tank model, Boltzmann machine, self-adaptive network, elastic network)
- Bayesian heuristics (global optimization, discrete optimization)
- Overlay (disturbance of data, disturbance of the parameters of a heuristic) [4].

Chapter 02: Vehicle Routing problems and their variants

2.1. Introduction

Vehicle routing problem is a common problem in operational research, it is a combinatorial optimization problem where we have area service consisting of a set of depots and a randomly distributed set of customers, also we have a fleet of identical vehicles of homogeneous or heterogenous capacity that the vehicle can carries [6].

The vehicle routing problem is extended with constraints as examples: if we consider the quantities of goods requirements then should we add the capacity constraints, if we consider the time assortment should we add the time window variant constraints,..... and so on [9].

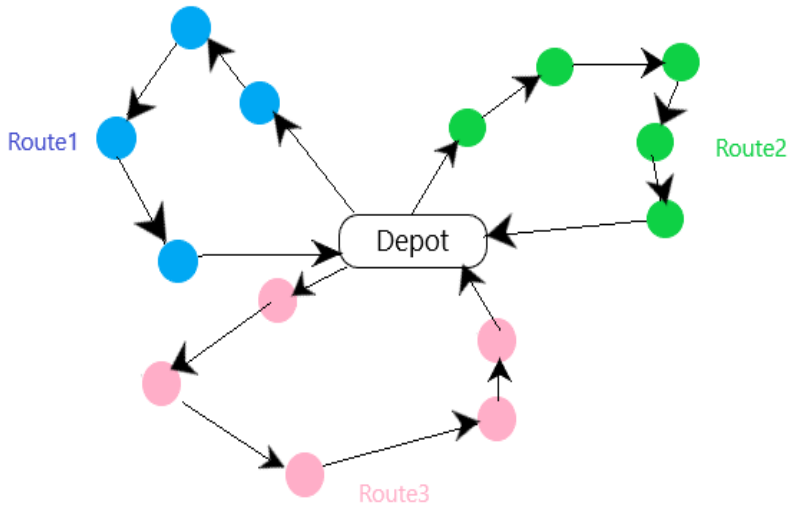

Figure 2. 1: model of an VRP with solution Single Depot VRP -with 3 vehicles [9].

2.2. Some vehicle routing problem variants

Table 2. 1: Vehicle routing problem variants

Some vehicle routing problem variants	**description**
Vehicle Routing Problem with Pickup and Delivery (VRPPD)	is a VRP in which the possibility that customers return some commodities is contemplated. So in VRPPD it's needed to take into account that the goods that customers return to the deliver vehicle must fit into it.
Vehicle Routing Problem with Time Windows (VRPTW)	can be defined as choosing routes for limited number of vehicles to serve a group of customers in the time windows(time interval). Each vehicle has a limited capacity. It starts from the depot and terminates at the depot. Each customer should be served exactly once.
VRP with Backhauls	in this variation, there are two subsets of customers: the first subset requires deliveries from the depot and the second subset requires goods to be picked up to be delivered to the depot.The total deliveries and the total pick-ups on each route must separately be less than the capacity of each vehicle
Dynamic VRP	is when the service requests are not completely known before the start of service, but they arrive during the distribution process(arrive dynamically) the routes have to be replanned at run time in order to include them
Capacitated Vehicle Routing Problem (CVRP)	is a VRP in which vehicles with limited carrying capacity need to pick up or deliver commodity at various locations. The commodity have a quantity, such as weight or volume, and the vehicles have a maximum capacity that they can carry. The problem is to pick up or deliver the commodity for the least cost, while never exceeding the capacity of the vehicles.

2.3. multi-depot vehicle routing problem MDVRP

2.3.1 Description of MDVRP:

Multi-depot vehicle routing problem (MDVRP) is one of the classical vehicle routing problems (VRP), where we multi-depot and multi-initial known customers each depot with their nearest customers (area of service), the MDVRP can be described as following: each depot with their customers represent a single depot vehicle routing problem, the optimization problems in MDVRP is allocated each customer to their nearest depot

The objective of the problem is to service all the customers of each depot and minimize travel distance.

We mention that there is a fleet of vehicles, and there are two cases to consider if the fleet of the homogenous vehicle or the heterogeneous fleet of the vehicle in terms of capacity to carry it.

also, we have two ways to solve the MDVRP: with a given number of vehicles or using enough vehicles.

In MDVRP vehicles are required to start from the depots visit all corresponding customers and return to the depots.

The case that we consider the capacity the problem becomes CMDVRP and we add the constraint formulation of capacity to the model [7].

2.3.2 Formulation of MDVRP:

The MDVRP can be formalized as follows. An undirected graph G = (V, E) is established to describe the mathematical model. In this model, $V = \{V_C, V_D\}$ represents the vertex set and $E = \{(v_I, v_J) | v_I, v_J \in V, i < j\}$ is the edge set.

$V_c = V_1, V_2, V_3, V_4 \ldots, V_n$ is the set of customers and $V_D = \{V_{n+1}, V_{n+2}, \ldots\ldots, V_{n+m}$ is the set of depots, in E, we get distance matrix $C = C_{i,j}$ by calculating the Euclidean

distance of customers V_i and $V_j$every customer V_i has a demand q_i and needs to be visited once by only one vehicle.[8]

there is also a fleet of K identical vehicles, each with capacity Q. In the mathematical formulation that follows, binary variable X_{ijk} is equal to 1 when vehicle k visits node j immediately after node i.[8]

Minimizing the total cost by the formula:

$$\sum_{i=1}^{n+m} \sum_{j=1}^{n+m} \sum_{k=1}^{K} CijXijk \qquad 2.(1)$$

For each vehicle k from k=1 to K starting from node i (i=1..n+m) ending in node j (=1..n+m) ,i≠j

Subject to :

$$\sum_{i=1}^{n+m} \sum_{k=1}^{K} X_{ijk} = 1 \ (j = 1,2,..,n); \qquad 2.(2)$$

(2) The first constraint which means each vehicle should visit each node one and only one time in each edge (path,cercle) in the complete solution of the problem

$$\sum_{j=1}^{n+m} \sum_{k=1}^{K} X_{ijk} = 1 \qquad (i = 1,2,\ldots.,n); \quad 2.(3)$$

(3) Guarantee that each customer is served by exactly one vehicle

Note that the value of k replaced in the formula is equal 1 all the time let's say it's boolean variable, for each vehicle k ,X_{ij} which is also a boolean variable which can be 1 or 0, so the result of all the formula it should be 1(if the customer is server by the vehicle ki or 0 (if not visited by the vehicle k_i [8] .

$$\sum_{i=1}^{n+m} \sum_{j=1}^{n+m} q_i * X_{ijk} \leq Q \ \ (k = 1,2,\ldots.,n); \qquad 2.(4)$$

Formula (4) it represents the constraint of vehicle capacity

it ensures that the summation of capacity requirement of subtours of the solution of the model don't exceed the vehicle capacity

As example: if we found in a VRP model a path that exceed the vehicle capacity then the other nodes in the path we will handle it by another new vehicle that make a new path which make a new path who collect the remaining nodes from previous path, or we will change the solution definitively

$$\sum_{i=n+1}^{n+m} \sum_{j=1}^{n} X_{ijk} \leq 1 \ (k = 1,2, \dots \dots, K) \qquad 2.\ (5)$$

(5) It guarantees that at more one vehicle it was come from node i to node j $i \in [0, n+m]$, $n\ represent\ set\ of\ customers\ m : represent\ set\ of\ depots$

$$\sum_{j=n+1}^{n+m} \sum_{i=1}^{n} X_{ij} \leq 1 \ (k = 1,2, \dots \dots, K) \qquad 2.(6)$$

(6) It guarantees that at more 1 vehicle it was leaved(visited) the node j

2.4. Conclusion

In this chapter, we have seen the various variants of vehicle routing problem, CVRP, VRPD, and MDVRP, Therefore, we conclude that the VRP can impose various variants and restrictions as we see we can represent by new formulation we add it in the model, this formulation should be checked in each search or let say generation of feasible solution (tour) in VRP, also we conclude that the objective of this problem or the main consideration problem in VRPs it to optimize the distance or the capacity of requirements in the routes by setting what should be Cost C_{ij}

And we have seen CMDVRP- multi depot vehicle routing problem considering capacity Description and mathematical formulation.

In the next chapter we see the algorithm Ant Colony Optimization solving vehicle routing problems and we detail in the Hybrid Ant Colony Optimization algorithm for MDVRP which is the main subject in this graduation project .

Chapter 03: Ant Colony Optimization Algorithm

3.1. Introduction

Ant colony optimization (ACO) is one of the most recent techniques for approximate optimization. The inspiring source of ACO algorithms are real ant colonies. More specifically, ACO is inspired by the ants' foraging behavior. At the core of this behavior is the indirect communication between the ants by means of chemical pheromone trails, which enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, for example, discrete optimization problems.

Depending on the point of view, ACO algorithms may belong to different classes of approximate algorithms. Seen from the artificial intelligence (AI) perspective, ACO algorithms are one of the most successful strands of swarm intelligence. The goal of swarm intelligence is the design of intelligent multi-agent systems by taking inspiration from the collective behavior of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or fish schools. Examples of "swarm intelligent" algorithms other than ACO are those for clustering and data mining inspired by ants' cemetery building behavior, those for dynamic task allocation inspired by the behavior of wasp colonies, and particle swarm optimization.

Seen from the operations research (OR) perspective, ACO algorithms belong to the class of metaheuristics**.** The term metaheuristic, first introduced in, derives from the composition of two Greek words. Heuristic derives from the verb heuriskein (ϱυρισκϱιν) which means "to find", while the suffix meta means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called modern heuristics. In addition to ACO, other algorithms such as evolutionary computation, iterated local search, simulated annealing, and tabu search, are often regarded as metaheuristics. For books and surveys on metaheuristics see [11]

3.2 The origins of ant colony optimization

Marco Dorigo and colleagues introduced the first ACO algorithms in the early 1990's . The development of these algorithms was inspired by the observation of ant colonies. Ants are social insects. They live in colonies and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ACO is the ants' foraging behavior, and in particular, how ants can find shortest paths between food sources and their nest. when searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants can smell pheromone. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown in that the indirect communication between the ants via pheromone trails—known as stigmergy —enables them to find shortest paths between their nest and food sources.

As a first step towards an algorithm for discrete optimization we present in the following a discretized and simplified model of the phenomenon, after presenting the model we will outline the differences between the model and the behavior of real ants. Our model consists of a graph G = (V,E), where V consists of two nodes, namely vs (representing the nest of the ants), and vd (representing the food source). Furthermore, E consists of two links, namely e1 and e2, between vs and vd . To e1 we assign a length of l1, and to e2 a length of l2 such that l2 > l1. In other words, e1 represents the short path between vs and vd , and e2 represents the long path. Real ants deposit pheromone on the paths on which they move. Thus, the chemical pheromone trails are modeled as follows. We introduce an artificial pheromone value τi for each of the two links ei , i = 1, 2. Such a value indicates the strength of the pheromone trail on the corresponding path. Finally, we introduce na artificial ants. Each ant behaves as follows: Starting from vs (i.e., the nest), an ant chooses with probability.

$$pi = \tau i / \tau 1 + \tau 2 \quad (i = 1, 2 \dots \dots n) \qquad 2.(1)$$

between path e1 and path e2 for reaching the food source vd . Obviously, if $\tau 1 > \tau 2$, the probability of choosing e1 is higher, and vice versa. For returning from vd to vs , an ant uses the same path as it chose to reach vd , and it changes the artificial pheromone value associated to the used edge. More in detail, having chosen edge ei an ant changes the artificial pheromone value τi as follows:

$$\tau i \leftarrow \tau i + Q / li \qquad 2.(2)$$

, where the positive constant Q is a parameter of the model. In other words, the amount of artificial pheromone that is added depends on the length of the chosen path: the shorter the path, the higher the amount of added pheromone. The foraging of an ant colony is in this model iteratively simulated as follows: At each step (or iteration) all the ants are initially placed in node vs . Then, each ant moves from vs to vd as outlined above. As mentioned in the caption of Fig. 1(d), in nature the deposited pheromone is subject to an evaporation over time. We simulate this pheromone evaporation in the artificial model as follows:

$$\tau i \leftarrow (1 - \rho) \cdot \tau i, i = 1, 2. \qquad 2.(3)$$

(3) The parameter $\rho \in (0, 1]$ is a parameter that regulates the pheromone evaporation. Finally, all ants conduct their return trip and reinforce their chosen path as outlined above. We implemented this system and conducted simulations with the following settings: l1 = 1, l2 = 2, Q = 1. The two pheromone values were initialized to 0.5 each. Note that in our artificial system we cannot start with artificial pheromone values of 0. This would lead to a division by 0 in Eq. (1). The results of our simulations are shown in Fig. 2. They clearly show that over time the artificial colony of ants converges to the short path, i.e., after some time all ants use the short path. In the case of 10 ants the random fluctuations are bigger than in the case of 100 ants ,this indicates that the shortest path finding capability of ant colonies results from a cooperation between the ants.[11]

3.3.1 A real ant

An ant is an insect that lives and works in a large colony of ants. Most ants don't have wings, and some of them have stingers.

Ants are related to both bees and wasps, and like them are social insects. Ant colonies can include anywhere from a few dozen to millions of ants, divided into jobs or castes. There isn't a continent in the world that

doesn't have ants living there. The Middle English word for ant was ampte, from the Old English æmette and a Germanic root. [14]

3.3.2Artificial ant:

An artificial ant simulates a real ant and a set of artificial ants develops mechanisms of cooperation and learning. Ant Colony Optimization (ACO) was developed by Dorigo et. al. in 1996 [1] and it was first used to solve combinatorial agents that imitate the behavior of real ants [11].

However, it should be noted that an artificial ant system has some differences in comparison with real ants which are as follows:

a) Artificial ants have memory.

b) They are not completely blind.

c) They follow a discrete time system.

The main idea is that when an ant has to select a path among several available paths, the ant chooses the one which is chosen more frequently by other ants in the past. Thus path with larger amount of pheromones is the shorter path and chosen by most of the ants. The Ant System works in two major steps:

a) Construction of the solution to the problem under consideration.

b) Updating the pheromone trails which may increase or decrease the amount of pheromone on certain paths.

3.3From Real to Artificial ant:

Ant colonies, and more generally social insect societies, are distributed systems that, in spite of the simplicity of their individuals, present a highly structured social organization. As a result of this organization, ant colonies can accomplish complex tasks that in some cases far exceed the individual capabilities of a single ant. The field of ‘‘ant algorithms’’ study models derived from the observation of real ants’ behavior, and uses these models as a source of inspiration for the design of novel algorithms for the solution of optimization and distributed control problems. The main idea is that the self-organizing principles which allow the highly coordinated behavior of real ants can be exploited to coordinate populations of artificial agents that collaborate to solve computational problems. Several different aspects of the behavior of ant colonies have inspired different kinds of ant algorithms. Examples are foraging, division of labor, brood sorting, and cooperative transport. In all these examples, ants coordinate their activities via incentives, a form of indirect communication mediated by modifications of the environment. For example, a foraging ant deposits a chemical on the ground which increases the probability that other ants will follow the same path. Biologists have shown that many colony-level behaviors observed in social insects can be explained via rather simple models in which only stigmergy communication is present. In other words, biologists have shown that it is often sufficient to consider stigmergic, indirect communication to explain how social insects can achieve self-organization. The idea behind ant algorithms is then to use a form of artificial stigmergy to coordinate societies of artificial agents. One of the most successful examples of ant algorithms is known as ‘‘ant colony optimization,’’ or ACO, and is the subject of this book. ACO is inspired by the foraging behavior of ant colonies, and targets discrete optimization problems. This introductory chapter describes how real ants have inspired the definition of artificial ants that can solve discrete optimization problems.[12]

3.4 Differences between real ants and artificial ants:

The main differences between the behavior of the real ants and the behavior of the artificial ants in our model are as follow:

(1) While real ants move in their environment in an asynchronous way, the artificial ants are synchronized, i.e., at each iteration of the simulated system, each of the artificial ants moves from the nest to the food source and follows the same path back.

(2) While real ants leave pheromone on the ground whenever they move, artificial ants only deposit artificial pheromone on their way back to the nest.

(3) The foraging behavior of real ants is based on an implicit evaluation of a solution (i.e., a path from the nest to the food source). By implicit solution evaluation we mean the fact that shorter paths will be completed earlier than longer ones, and therefore they will receive pheromone reinforcement more quickly. In contrast, the artificial ants evaluate a solution with respect to some quality measure which is used to determine the strength of the pheromone reinforcement that the ants perform during their return trip to the nest.[11][13]

Natural Ant colony	Artificial Ant colony
Ant	**Agent**
Ant colony	**Sites of Ants/Iterations**
Pheromone	**Diversity mechanism**
Path	**Solution**
Evaporation	**Pheromone update**

Table3.1 Analogy between Natural and Artificial Ants [15]

3.5 ACO (ant colony optimization)

3.5.1 Description:

Ant colony optimization is a search technique used in computing to find near optimal solutions to discrete optimization problems.

ACO is swarm intelligence inspired from the way that ants indirectly communicate directions to each other.[16]

The most interesting aspect of the collaborative behavior of ant species is their ability to find the shortest paths between the ant's nest and food sources.

Finding the shortest path between their nest and a food source by chase pheromone trails exposed by other ants, more instance trails the higher probability that an ant will follow it and thus enrich the trail with its own pheromone.

One of the properties of pheromone is that it evaporates over time, and the ant sense the pheromone trail for travelling over the nest or searching the food.

the pheromone trail of the longest paths evaporates and that it because it takes more time from the shortest path, so logically the shortest path will have a higher density of pheromone trail, also because the pheromone update for each ant took the path (deposited & evaporated pheromone). [17]

3.5.2 Biological ACO ant colony optimization:

Let's see an example of this. let consider there are two paths to reach the food from the colony. At first, there is no pheromone on the ground. So, the probability of choosing these two paths is equal that means 50%.50% Let consider two ants choose two different paths to reach the food as the probability of choosing these paths is fifty-fifty.

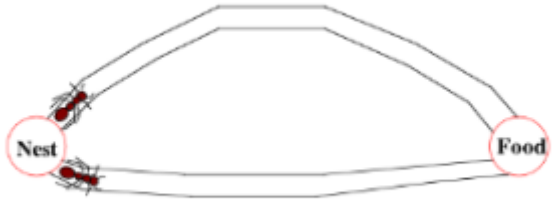

figure3. 1Step 1 in biological ACO [8]

The distances of these two paths are different. ant following the shorter path will reach the food earlier than the other.

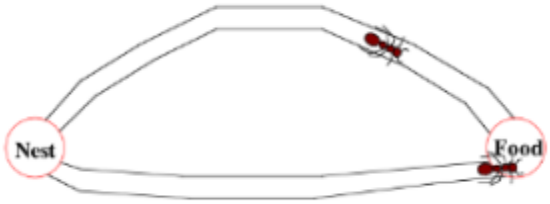

figure3. 2 Step2 in biological ACO [8]

After finding food, it carries some food with itself and returns to the colony. when it tracking the returning path it deposits pheromone on the ground. The ant following the shorter path will reach the colony earlier.

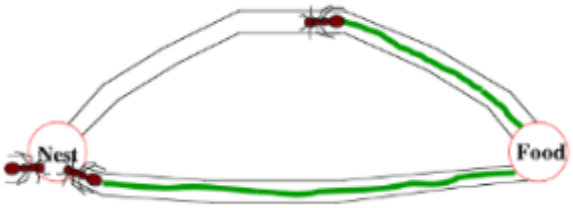

figure3. 3: Step3 in biological ACO [8]

When the third ant wants to go out for searching food it will follow the path having shorter distance based on the pheromone level on the ground. As a shorter path has more pheromones than the longer, the third ant will follow the path having more pheromones.

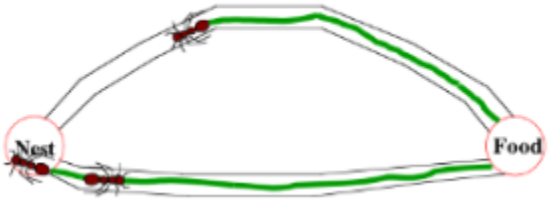

figure3. 4: Step4 in biological ACO [8]

By the time the ant following the longer path returned to the colony, Then when another ant tries to reach the destination(food) from the colony if it find that each path has the same pheromone level it can not differentiate between the densities of paths, it randomly chooses one of them. Let consider it choose the above one(in the picture located below)

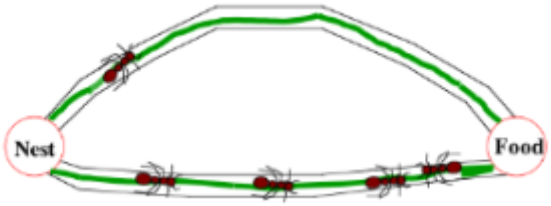

figure3. 5: Step5 in biological ACO [8]

Repeating this process again and again, after some time, the shorter path has a more pheromone level than others and has a higher probability to follow the path, and all ants next time will follow the shorter path.

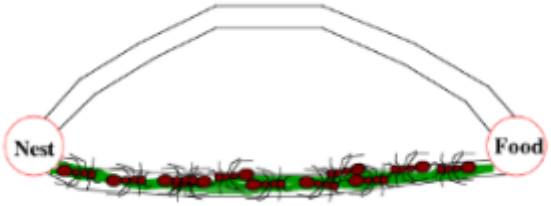

figure3. 6: Step6 in biological ACO [8]

For solving different problems with ACO, there are three different proposed version of Ant-System AS:

Ant Density & Ant Quantity: Pheromone is updated in each movement of an ant from one location to another.

Ant Cycle: Pheromone is updated after all ants completed their tour.

Pheromone Update . [8]

3.5.3 metaheuristic ACO algorithm and formulation:

An artificial ant is made for finding the optimal solution. In the first step of solving a problem, each ant generates a solution. In the second step, paths found by different ants are compared. And in the third step, the paths value or pheromone is updated.

```
procedure ACO_MetaHeuristic
while not_termination do
    generateSolutions()
    daemonActions()
    pheromoneUpdate()
    repeat()
end procedure
```

Pseudo Code: ACO metaheuristic algorithm in pseudo code [19]

3.5.3.1Basic ACO formulation

Each ant needs to construct a solution to move through the graph. To select the next edge in its tour, an ant will consider the length of each edge available from its current position, as well as the corresponding pheromone level. At each step of the algorithm, each ant moves from a state x to state, corresponding to a more complete intermediate solution. Thus, each ant computes a set $A_k(x)$ $A$of feasible expansions to its current state in each iteration, and moves to one of these in probability. For ant $\boldsymbol{Kth}$, the probability $\boldsymbol{P_{xy}^k}$ of moving from state $\boldsymbol{x}$ to state $\boldsymbol{y}$ depends on the combination of two values, the attractiveness $\boldsymbol{\eta_{xy}}$of the move, as computed by some heuristic indicating the a priori desirability of that move and the trail level $\boldsymbol{\tau_{xy}}$ of the move, indicating how proficient it has been in the past to make that particular move, the trail level represents a posteriori indication of the desirability of that move [19]

In general, the kth ant moves from state x to state y with probability:

$$P_{xy}^{k} = \frac{(\tau_{xy})^{\alpha}(\eta_{xy})^{\beta}}{\sum_{z\in\, allowed\ x}(\tau_{xz})^{\alpha}(\eta_{xz})^{\beta}} \qquad 2.(4)$$

where τ_{xy} is the amount of pheromone deposited for transition from state x to y , $0 \leq \alpha$ is a parameter to control the influence of τ_{xy}, η_{xy} is the desirability of state transition xy (a priori **knowledge, typically** $\frac{1}{d_{xy}}$, where d is the distance (between state x and y**), and** $\beta \geq 1$ is a parameter to control the influence of η_{xy}.

τ_{xz} and η_{xz}represent the trail level and attractiveness for other possible state transitions.

3.5.3.1.1) Pheromone update:

Trails are usually updated when all ants have completed their solution, increasing or decreasing the level of trails corresponding to moves that were part of "good" or "bad" solutions, respectively.

An example of a global pheromone updating rule **is:**

$$\tau_{xy_{new}} \longleftarrow (1-\rho)\tau_{xy_{old}} + \sum_{k}^{m} \Delta\tau_{xy}^{k} \qquad 2.(5)$$

Where τ_{xy} is amount of pheromone deposited for a state transition xy, ρ is the pheromone evaporation coefficient**,** m is the number of ants and $\Delta\tau_{xy}^{k}$is the amount of pheromone deposited by kth **ant,** typically given for TSP problem (with moves corresponding to arcs of the graph) **by:**

$$\Delta\tau_{xy}^{k} = \begin{cases} \frac{Q}{L_k} & if\ ant\ k\ uses\ curve\ xy\ in\ its\ tour \\ 0 & other\ wise \end{cases} \qquad 2.(6)$$

Where $\boldsymbol{L_k}$ the cost of kth ant's tour (typically length) and $\boldsymbol{Q}$ is a constant

3.5.4 Some problems resolved with ant colony optimization ACO

The ACO meta-heuristic has been applied to various different combinatorial optimization problems with a large number of successful implementations. Applications to static combinatorial optimization issues include the following:

- **travelling salesman Problem**, where a salesman must find the shortest route by which he can visit a given number of cities, each city exactìy once.
- **Quadratic Assignment Problem,** the problem of assigning n facilities to n locations so that the costs of the assignment are minimized.
- **Job-Shop Scheduling Problem**, In order to avoid having two jobs processed simultaneously on the same machine and to reduce the total time required to complete all operations, a given set of machines and a given set of job operations must be assigned to time intervals.
- **Vehicle Routing Problem,** the objective is to find minimum cost vehicle Routes such thât:
  - Every customer is visited exactly once by exactly one vehicle;
  - For every vehicle the total demand does not exceed the vehicle capacity;
  - The total tour length of each ræhicle does not exceed a given limit;
  - Every vehicle starts and ends its tour at the same position.[27]

3.6 Conclusion:

Ant algorithms are inspired by the self-regulating search behavior of natural ants, which show amazing resilience, adaptability and scalability despite being based on a set of simple mechanisms.

The ant colony optimization algorithm is a metaheuristic algorithm used in many different types of harmonic optimization problems, difficult NP problems, ..., such as subsets, vehicle routing, scheduling...

The ACO algorithm adapts to different problems based on datasets and some modifications in the ant system, as well as adding and modifying the limitations and limitations

of the algorithms, but the main idea of the algorithm remains the same for multiple extensions of the ACO algorithm.

Chapter 04: Hybrid Ant Colony Optimization for Multi depot Vehicle Routing

4.1 A Hybrid Ant Colony Algorithm for MDVRP [1]

4.1.1 Hybrid Ant Colony Optimization

In this study we applied the nearest distance clustering approach to multi depot vehicle routing problem to allocate each customer to their nearest depot, after the clustering process we give the results of clustering to the ACO algorithm solving multi depot vehicle routing problem after ACO find the solution for each submodel (VRP)we move to the optimization process and we apply the local interchange technique (2-opt algorithm) to obtain better approximative solutions from the ant colony algorithm for MDVRP.

4.1.2. The Nearest Distance Cluster Algorithm.

The nearest distance clustering approach is a technique used to divide the whole area of delivery service which contain multi-depot and multi-knowed customer into multi-area service each depot with their nearest customers.[23]

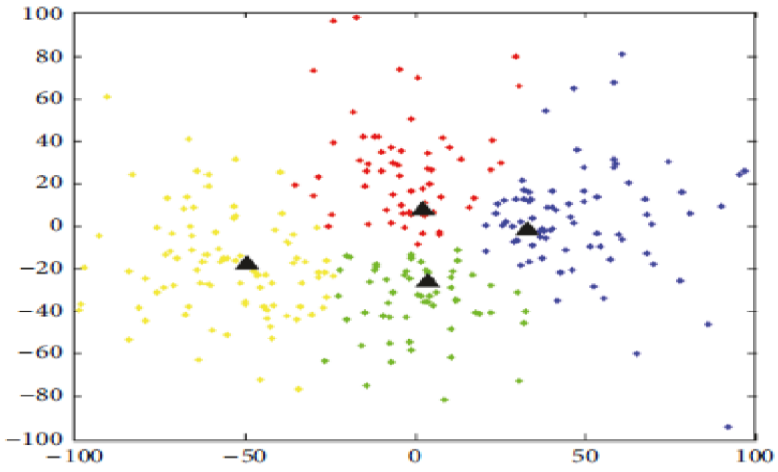

figure 4. 1: An example of the nearest distance cluster algorithm. [23]

4.1.3 Generate Initial Solutions. In ACO

After generating all the feasible solutions to the problem, we give it to the ACO algorithm to search for the best solution tours and costs, each ant visits all the vertexes (customers) of all the feasible solutions once at more. The complete routes that ants have passed are initial solutions. The ants will decide to select the next customer in the feasible solutions by formula 4. (1).

$$p_{i,j}(k) = \begin{cases} \frac{\tau(i,j)^{\alpha} \times \eta(i,j)^{\beta}}{\sum I \notin tabu\ \tau(i,j)^{\alpha} \times \eta(i,j)^{\beta}} & j \notin tabu\ k \\ 0 & otherwise \end{cases} \quad 4.\ (1)$$

The formula 4.1 represents the probability that the ant (k) select the node j as the next customer to visit it, the probability is equal to $\tau(i,j)^{\alpha}$ which is the density of pheromone in the edge(i,j) to the power of constant (α) which is the relative influence of density of pheromone multiple $\eta(i,j)^{\beta}$ which is the visibility of edge(I,j) to the power of β: the relative influence of visibility of the edge [23].

4.2 Optimization process [1]

4.2.1. Local Interchange Operation 2-opt

It's local optimization heuristic technique and most common of its algorithm it's 2-opt algorithm, it consists in breaking all possible pair of adjacent edges in tour and reconnecting them differently in the purpose of obtain minimal distance of the tour, if the cost of the new tour is minimal than the recent then it update it to new form, otherwise it hold the same tour [23].

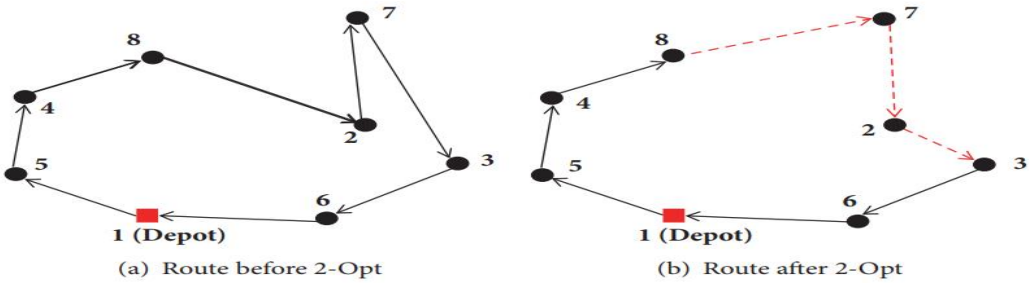

figure 4. 2: The demo of 2-Opt-operation [23]

4.3. Update of Pheromone Information.

pheromone updating is the most important thing that assumes the quality of the solutions of ACO algorithm, there are two kinds of systems of pheromone updating

Ant Cycle: Pheromone is updated after all ants completed their tour.then

Ant Density & Ant Quantity: Pheromone is updated in each movement of an ant from one location to another

$$\tau_{i,j}^{new} = \rho \times \tau_{i,j}^{old} + \sum_{k}^{K} \Delta\tau_{i,j}^{k} \quad \rho \in (0.1) \quad 4.(4)$$

In formula (4), $\tau_{i,j}$new : is the value of new(updated) density of pheromone

$\rho \times \tau_{i,j}^{old}$ it represent the evaporation process,+$\sum_{k}^{K} \Delta\tau_{i,j}^{k}$: represent the recompense process, so the pheromone updating consists on this two primary process[1] .

4.6. Conclusions

In this chapter we presented some of the big points of our work in the theoretical part, which also represents what we have achieved in the practical aspect of the graduation project

first of all we have seen the nearest distance clustering approach which consists of the allocation and division of the area service into submodel of VRP problems, also we have seen and how the ACO algorithm generate the initial solution.

secondly, we see the optimization process and we have explained the 2-opt algorithm, thirdly and finally, the update pheromone which is the most important process in the ACO algorithm for the quality of the solution

Chapter 05: Implementation and results

5.1. Introduction

This chapter is dedicated to the contribution of our final graduation project, which consists of Hybrid Ant Colony Optimization for multi depot vehicle routing problem

The general plan in this chapter is as follows:

1- Create a model (Algeria problem) that contains the coordinates of warehouses and customers using these coordinates.

2-apply the nearest specific distance algorithm in order to allocate each customer to his nearest warehouse with the quantity of his goods.

3- ACO modification for VRPTW solution to deal with CMDVRP problem

4- Apply 2opt to improve results

During this study, we used the MATLAB R2016a version to perform the experiments.

5.2.MATLAB Programming language and Environment

Uses for MATLAB include matrix calculations, developing and running algorithms, creating user interfaces (UI) and data visualization. The multi-Paradigm numerical computing environment allows developers to interface with programs developed in different languages, which makes it possible to harness the unique strengths of each language for various purposes.

MATLAB is used by engineers and scientists in many fields such as image and signal processing, communications, control systems for industry, smart grid design, robotics as well as computational finance.

Cleve Moler, a professor of Computer Science at the University of New Mexico, created MATLAB in the 1970s to help his students. MATLAB's commercial potential was identified by visiting engineer Jack little in 1983. Moler, Little and Steve Bangart founded MathWorks and rewrote MATLAB in C under the auspices of their new company in 1984. [24]

MATLAB is an abbreviation of Matrix Laboratory. The current version; written in C by MathWorks Inc; exists in the professional version and student version, its availability is ensured on several platforms.

MATLAB is a powerful, comprehensive, and easy-to-use environment for scientific computing. It allows to perform numerical simulations based on numerical analysis algorithms or generic algorithms.

MATLAB is considered one of the best programming languages (C or Fortran), it has the following particularities compared to these languages:

– Easy programming;

– Continuity among real and complex integer values;

– Extended range of numbers and their precision;

– Very comprehensive mathematical library;

– GUI tool that includes GUI functions and utilities;

– Possibility of linking with other classic programming languages (C or Fortran). [25]

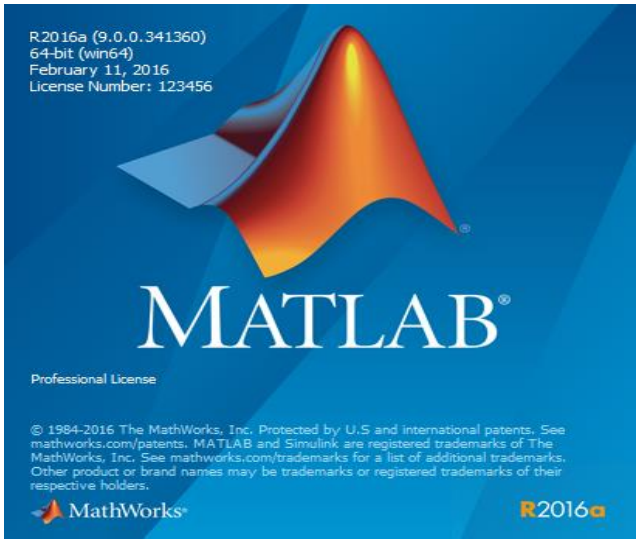

figure 5. 1: MATLAB R2016a

5.3 Execution & Results:

In this section we test the implemented algorithm Hybrid ant colony optimization, through multiple series of executions, through it we change the values of different parameters (variables) and see the influence of each parameter on the results, also the best values of parameters for the algorithm to obtain the better-optimized solutions.

5.3.1. The effect of changing the relative influence of the pheromone trails α on the results

We went to take 3 different values of α from 3 different fields, note that α ranges o from [0,1] divide the range into three fields to evaluate the algorithm with these fields. [0,0.33], [0.33, 0.66], [0.66, 0.99]

5.3.1.a)1st first set of α:

fix the values of the variables, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and put

α = 0.1 $\in$[0,0.33]

Table 5. 1: statistics on the executions of 1st value α=0.1

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	23.7898
2	1010.1565	23.2401
3	1010.1565	24.6054
4	1010.1565	25.947
5	1010.1565	24.704
6	1010.1565	24.4916
7	1010.1565	22.058
8	1010.1565	26.0759
9	1010.1565	23.4926
10	1010.1565	24.9774
Mean Sol	**1010.1565**	**24.33818**

5.3.1.b)2nd second set of α:

We fix the values of the variables, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and put

α = 0.5 ∈ [0.33,0.66] .

Table 5. 2: statistics on the executions of the 2nd value α=0.5

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	17.8113
2	1010.1565	18.3991
3	1010.1565	18.5159
4	1010.1565	18.1879
5	1010.1565	17.0385
6	1010.1565	19.0805
7	1010.1565	16.8803
8	1010.1565	16.4054
9	1010.1565	17.7371
10	1010.1565	16.8966
Mean Sol	**1010.1565**	**17.69526**

5.3.1.c)3rd third set value of α:

fixing the values of the variables, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and change

α = 0.9 ∈ [0.33,0.66]

Table 5. 3: statistics on the executions of the 3rd value =0.9

Executios	Total cost of the overall solution	Total Time of Execution
1	1010.1565	16.6367
2	1010.1565	17.3166
3	1010.1565	16.7008
4	1010.1565	16.137
5	1010.1565	20.1036
6	1010.1565	18.7094
7	1010.1565	16.9421
8	1010.1565	19.3519
9	1010.1565	18.2672
10	1010.1565	18.264
Mean Sol	**1010.1565**	**17.84293**

Comment: the results stay the same in the 3 series of experiment .

Result: the variation of α (relative influence of the pheromone trail) doesn't affect on the obtained solution from this algorithm , the solution is the same as is shown in the following screenshots :

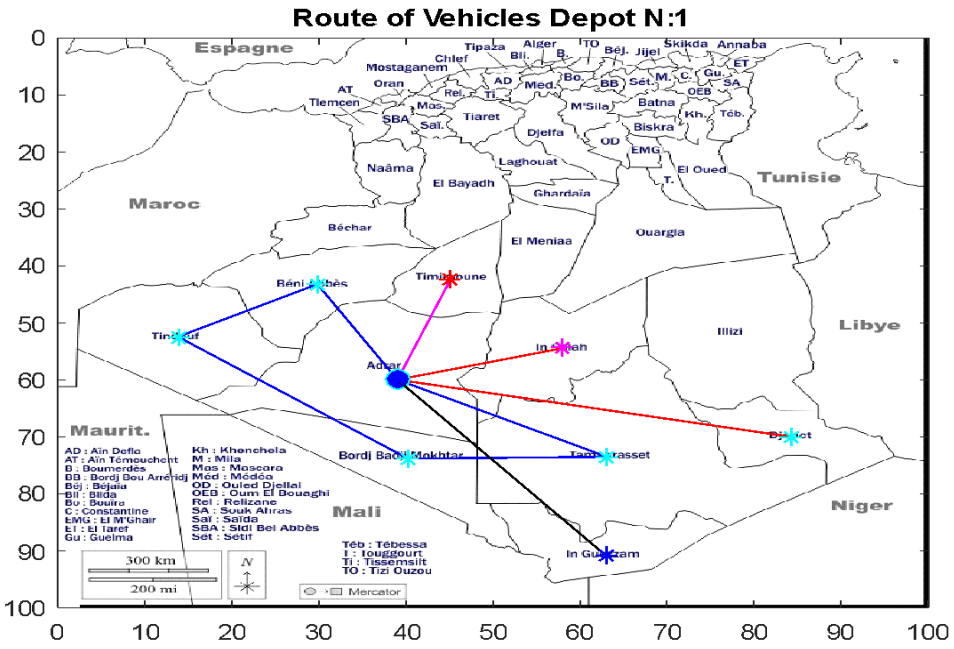

Figure 5.2: Depot 1 paths, the best solution obtained from the execution's series of α

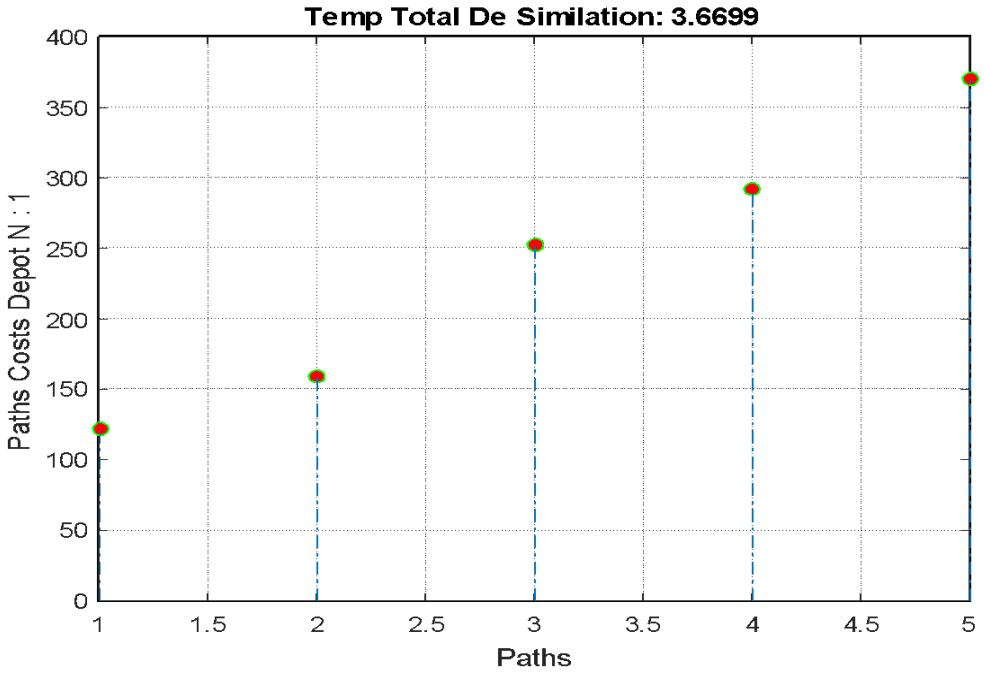

Figure 5.3: cost paths of depot 1 of the best solution obtained from the execution's series of α

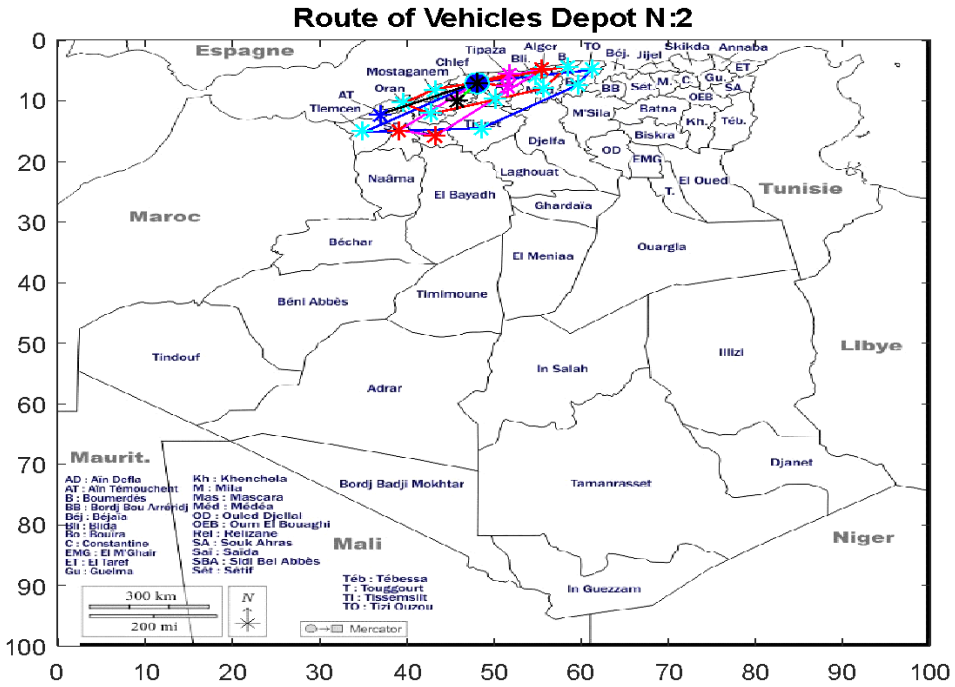

Figure 5.4: Depot 2 paths ,the best solution obtained from the execution's series of α

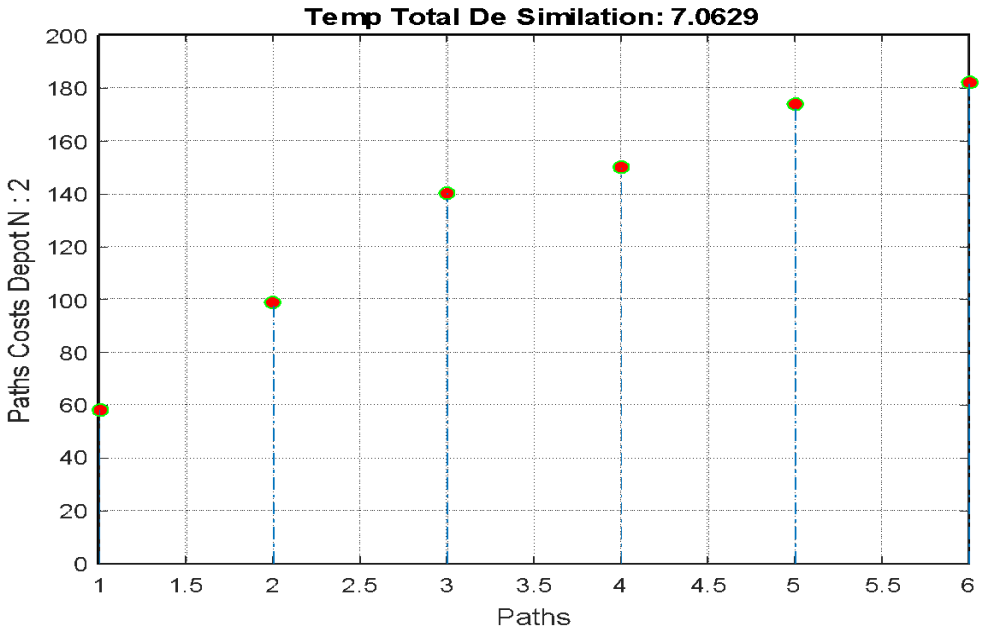

Figure 5.5: Depot 2 path costs of the best solution obtained from the execution's series of α

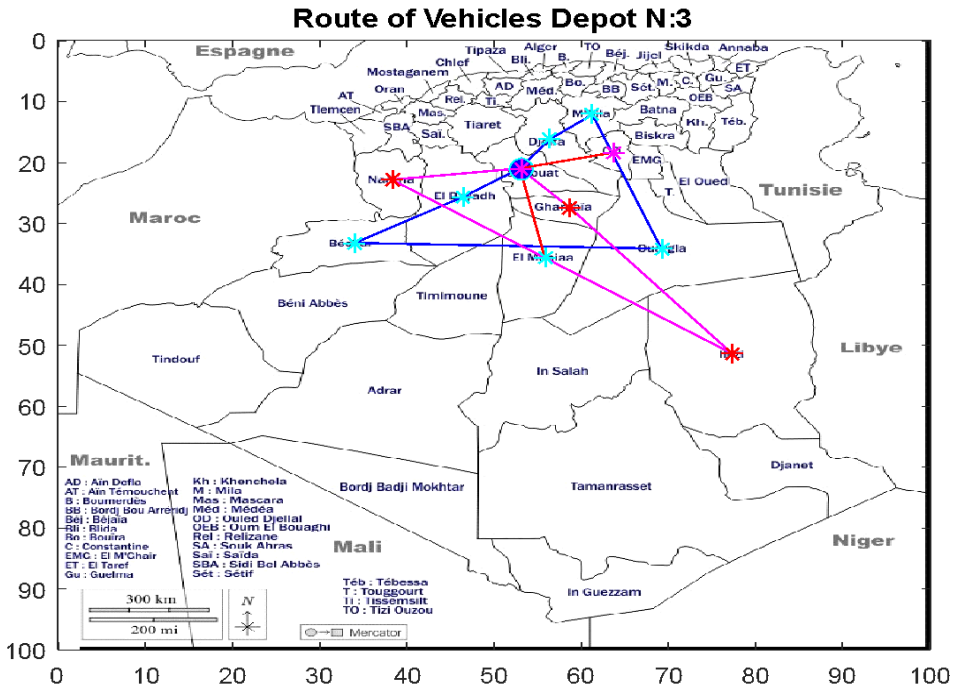

Figure 5.6: Depot 3 paths of the best solution obtained from the execution's series of α

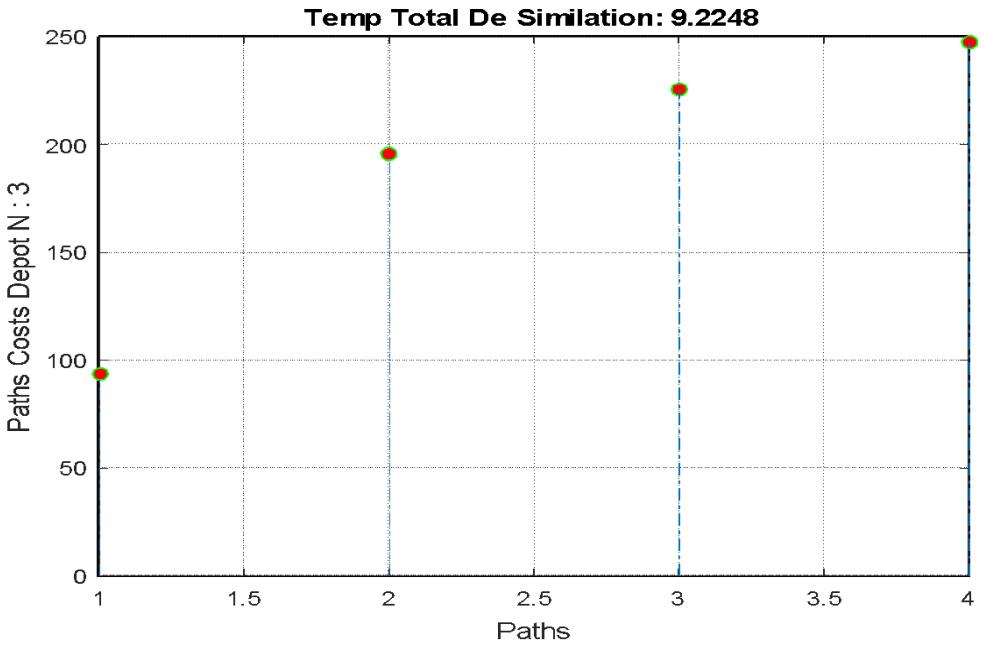

figure 5.7: Depot 3 costs paths of the best solution obtained from the execution's series of α

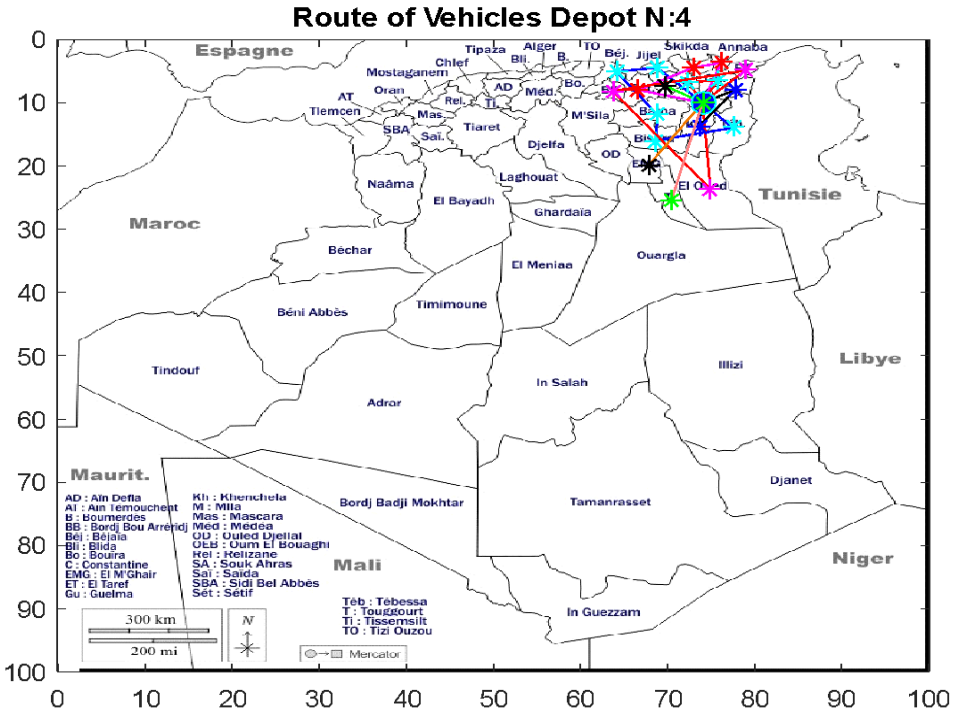

figure 5.8: Depot 4 paths of the best solution obtained from the execution's series of α

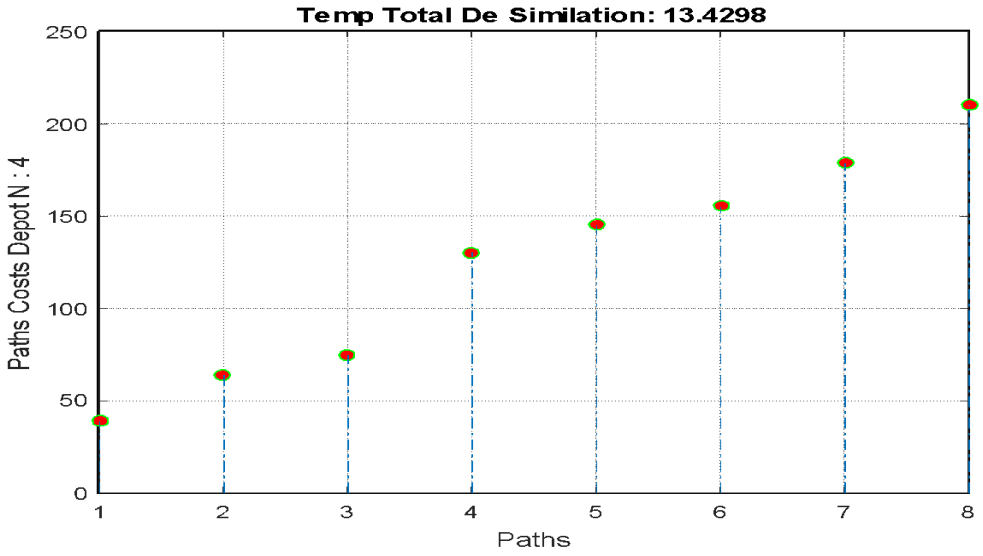

figure 5.9: Depot 4 costs paths of the best solution obtained from the execution's series of α

5.3.2. The effect of changing the visibility of edges β on the results

we gone take 3 different values of β from 3 different fields, note that α it ranges [0,1]

divide the range into three fields to evaluate it [0,0.33], [0.33, 0.66], [0.66, 0.99]

5.3.1.a)1st first set of β :

fixing the values of the variables, α = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and change

β = 0.1 $\in$[0,0.33]

Table 5. 4: statistics on the executions of the 1st value $=0.1$

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.2697
2	1010.1565	16.1216
3	1010.1565	16.7182
4	1010.1565	16.7332
5	1010.1565	17.6527
6	1010.1565	18.366
7	1010.1565	19.3974
8	1010.1565	15.5569
9	1010.1565	18.3908
10	1010.1565	17.8099
Mean	**1010.1565**	**17.20164**

5.3.1.b)2nd second set of β:

fixing the values of the variables, α = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and change

$\beta = 0.5 \in [0.33,0.66]$

Table 5. 5: statistics on the executions of the 2nd value $\beta = 0.5$

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.5975
2	1010.1565	17.0728
3	1010.1565	15.1839
4	1010.1565	23.5654
5	1010.1565	17.4862
6	1010.1565	16.2271
7	1010.1565	19.4945
8	1010.1565	19.6949
9	1010.1565	14.9137
10	1010.1565	16.7355
Mean	**1010.1565**	**17.59715**

5.3.1.c)3rd third set value of β:

fixing the values of the variables, α = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and change

β = 0.9 ∈[0.33,0.66]

Table 5. 6: statistics on the executions of 3rd value β =0.9

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	16.8437
2	1010.1565	16.5081
3	1010.1565	17.8804
4	1010.1565	16.5376
5	1010.1565	17.2613
6	1010.1565	18.3058
7	1010.1565	18.4801
8	1010.1565	19.0081
9	1010.1565	17.2116
10	1010.1565	15.4064
Mean	**1010.1565**	17.34431

5.3.3. The effect of changing the evaporation rate value on the results:

we will test the algorithm through series of execution in 3 fields of ρ , rho range from [0,1]

3 fields :[0,0.33], [0,0.66], [0,1] and see the effect of the evaporation rate on the solutions

5.3.1.a)1st first set of ρ:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change ρ=0.33∈[0,0.33]

Table 5. 7: statistics of the executions1st value ρ=0.33

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	18.0996
2	1010.1565	14.7027
3	1010.1565	15.2463
4	1010.1565	15.9064
5	1010.1565	14.9029
6	1010.1565	15.0762
7	1010.1565	16.1177
8	1010.1565	14.9721
9	1010.1565	13.8988
10	1010.1565	15.0216
Mean	**1010.1565**	**15.39443**

5.3.1.b) Second 2nd set of ρ:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, tau0=10*Q/(n_vertices*mean(model(idd).D(:)))

capacity= 100, Q = 1 ,and change ρ=0.66∈[0.33,0.66]:

Table 5. 8: statistics on the executions of the 2nd value ρ=0.66

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.0909
2	1010.1565	15.2307
3	1010.1565	14.6856
4	1010.1565	16.0694
5	1010.1565	15.2471
6	1010.1565	14.8648
7	1010.1565	14.7766
8	1010.1565	14.86
9	1010.1565	14.336
10	1010.1565	14.0195
Mean	**1010.1565**	**14.8122**

5.3.1.c) Third 3rd set of ρ:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, tau0=10*Q/(n_vertices*mean(model(idd).D(:)))

and change Q = 1 ,ρ=0.99∈[0.66,1]

Table 5. 9: statistics on the executions of the 3rd value =0.99

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.6629s
2	1010.1565	14.6609s
3	1010.1565	14.0819s
4	1010.1565	14.4726s
5	1010.1565	14.7744s
6	1010.1565	23.804s
7	1010.1565	14.4657s
8	1010.1565	16.882s
9	1010.1565	16.8814s
10	1010.1565	15.2762s
Mean	**1010.157**	**16.0962s**

The Comment: the results stays the same in the three sets of evaporation rate ρ ∈[0,0.33] || [0,0.66] || [0,1] , as it shown in the following screenshots :

5.3.4. The effect of changing the control variable pf pheromone generation Q value on the results

we will test 5 fields of Q

[0,0.33], [0,0.66], [0,1], [1,1,5], [1.5,2]

5.4.1.a)1st value of Q:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, tau0=10*Q/(n_vertices*mean(model(idd).D(:)))and change Q=0.33∈[0,0.33]

Table 5. 10: statistics of the executions of 1st value of $Q=0.33$

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	21.2079s
2	1010.1565	18.1624s
3	1010.1565	25.8393s
4	1010.1565	15.3289s
5	1010.1565	14.4834s
6	1010.1565	15.0083s
7	1010.1565	17.4731s
8	1010.1565	15.9019s
9	1010.1565	14.455s
10	1010.1565	12.8908s
Mean	**1010.1565**	**17.0751s**

5.4.1.b) 2nd value of Q:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change Q = 0.66∈ [0.33,0.66]

Table 5. 11: statistics on the executions the 2nd value of Q =0.66

Executions	Total cost of the overall solution	Total Time of Esxecution
1	1010.1565	15.5812s
2	1010.1565	14.2151s
3	1010.1565	14.9843s
4	1010.1565	12.8879s
5	1010.1565	14.4614s
6	1010.1565	16.8765s
7	1010.1565	15.3022s
8	1010.1565	15.4453s
9	1010.1565	16.2801s
10	1010.1565	15.2497s
Mean	**1010.157**	**15.12837s**

5.3.1.c)3rd third value of Q:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change Q = 0.99∈ [0.66,1]

Table 5. 12: statistics on the executions 3rd set of Q=0.99

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	16.7151s
2	1010.1565	15.0589s
3	1010.1565	13.5972s
4	1010.1565	13.9565s
5	1010.1565	19.988s
6	1010.1565	15.5746s
7	1010.1565	16.2334s
8	1010.1565	15.0293s
9	1010.1565	13.9991s
10	1010.1565	13.6982s
Mean	**1010.157**	**15.38503s**

5.3.1.4)4th fourth value of Q:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

capacity = 100, and change Q=1.5∈[1,1.5]

Table 5. 13: statistics on the executions of the 4th value $Q=1.5$

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	34.8313
2	1010.1565	28.9737
3	1010.1565	24.8756
4	1010.1565	15.7036
5	1010.1565	19.4109
6	1010.1565	19.3153
7	1010.1565	14.1405
8	1010.1565	16.3556
9	1010.1565	17.2548
10	1010.1565	16.0877
Mean	**1010.157**	**20.6949**

5.3.1.5)5th fifth value of Q:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

capacity = 100, and change Q=1.99∈[1,2]

Table 5. 14: statistics on the executions 5th value $Q=1.99$

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.5704
2	1010.1565	17.6127
3	1010.1565	15.1352
4	1010.1565	14.1629
5	1010.1565	15.355
6	1010.1565	16.1742
7	1010.1565	15.1297
8	1010.1565	14.6963
9	1010.1565	15.7565
10	1010.1565	14.5887
Mean	**1010.157**	**15.41816**

The comment: the results stay the same when we change the value of generation pheromone

In the five fields [0,0.33], [0,0.66], [0,1], [1,1,5], [1.5,2] ,as shown in bellow screen shoots of α series figure:5.20 , 5.21 , 5.22, ……..,5.27.

5.3.5. The effect of changing the initial pheromone τ_0 on the results:

we will 3 test values of Initial pheromone τ_0 0.1, 0.5 , 1

5.3.6.a)1st first value of τ_0:

Fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))), and change τ_0=0.1

Table 5. 15: statistics on the executions of the 1st value τ_0=0.1

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	16.7782
2	1010.1565	16.5702
3	1010.1565	16.5426
4	1010.1565	14.3073
5	1010.1565	15.5371
6	1010.1565	16.7037
7	1010.1565	14.5649
8	1010.1565	14.2499
9	1010.1565	14.7153
10	**1010.1565**	**14.1418**
Mean	**1010.157**	15.4111

5.3.6.b)2nd value of τ_0:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

capacity = 100, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change τ_0=0.5

Table 5. 16: statistics on the executions of the 2nd value of τ_0=0.5

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	15.3326
2	1010.1565	15.4832
3	1010.1565	15.8016
4	1010.1565	15.6283
5	1010.1565	15.0656
6	1010.1565	14.7712
7	1010.1565	16.1916
8	1010.1565	16.2755
9	1010.1565	15.1233
10	1010.1565	17.3567
Mean	**1010.157**	**15.70296**

5.3.6.c)3rd value of τ_0:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))) and change τ_0=0.9

Table 5. 17: statistics on the executions of the 3rd set of τ_0=0.9

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	14.7282
2	1010.1565	16.2033
3	1010.1565	18.3834
4	1010.1565	14.9507
5	1010.1565	15.1325
6	1010.1565	17.1709
7	1010.1565	16.8718
8	1010.1565	22.8703
9	1010.1565	18.3755
10	1010.1565	15.2013
Mean	**1010.157**	**16.98879**

The comment: the results stay the same for the model, when we change the In

the initial pheromone doesn't affect in the solution of this model

5.3.6. The effect of changing the capacity of vehicle on the results:

In this section we will test the algorithm with 3 different of capacity 50,100,150

Note that the requirements demands quantity range from 0-72

5.3.6.a)1st value of capacity <= maxdemand(72):

Fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

capacity = 100, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change to capacity=50

```
Entrer le nombre de depots a avoir4
The vehicle does not have the capacity to satisfy clients demand
Subscript indices must either be real positive integers or logicals.

Error in aco (line 182)
  TOSC=TOSC+Totalcost_route(end);
```

Figure 5.10: the error of the execution of the algorithm with capacity =50

The comment: Error in the code, note: that the used value of capacity of vehicle is 50 and the quantity demands range from 0 to 72 .

5.4.6.b)2nd value of capacity=150 > maxdemand=72:

fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change to capacity=150

Table 5.18: statistics on the executions of the 2nd value of *capacity* =150

Executions	Total cost of the overall solution	Total Time of Execution
1	849.4119	14.363
2	849.4119	14.6398
3	849.4119	17.4118
4	849.4119	14.973
5	849.4119	14.9699
6	849.4119	14.6453
7	849.4119	14.8868
8	849.4119	15.0072
9	849.4119	15.7647
10	849.4119	16.0156
Mean	**849.4119**	**15.26771**

comment: we notice change in routes of solution of the model, it optimized from the fixed value of capacity which is 100, here we get solution with Total cost =849.4119

Which is optimized with 15 %, where we add about 50 % of the capacity of vehicle to capacity of the vehicle capacity=150.

5.4.6.c)3rd value of capacity=250 :

Fixing the values of the variables α = 1, β = 1, maxIt = 50, nAnt = 40, ρ = 0.65,

, Q=1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),and change to capacity=250

Table 5.19: statistics of the executions of the 3rd value of *capacity* =250

Executions	Total cost of the overall solution	Total Time of Execution
1	680.6695	13.2609
2	680.6695	14.4422
3	680.6695	13.8766
4	680.6695	14.172
5	680.6695	15.5273
6	680.6695	13.79
7	680.6695	18.6699
8	680.6695	17.1602
9	680.6695	15.3675
10	680.6695	13.6753
Mean	**680.6695**	**14.99419**

The comment: we notice change in routes of solution of the model, it optimized from the fixed value of capacity which is 100, here we get solution with Total cost =849.4119

Which is optimized with 32 %, where where we add about 150 % of the capacity of vehicle to capacity of the vehicle.

Result: we conclude that the capacity of the vehicle effect on the solutions, When we have more capacity, we have improved solutions and better short routes, but the rule stops when the capacity of the vehicle exceeds the quantity of the total quantity of goods for warehouse customers.

The best solution gated from this variable is the following screenshots while that the capacity is greater than the total quantity of demands of all customer of depot, capacity >=700:

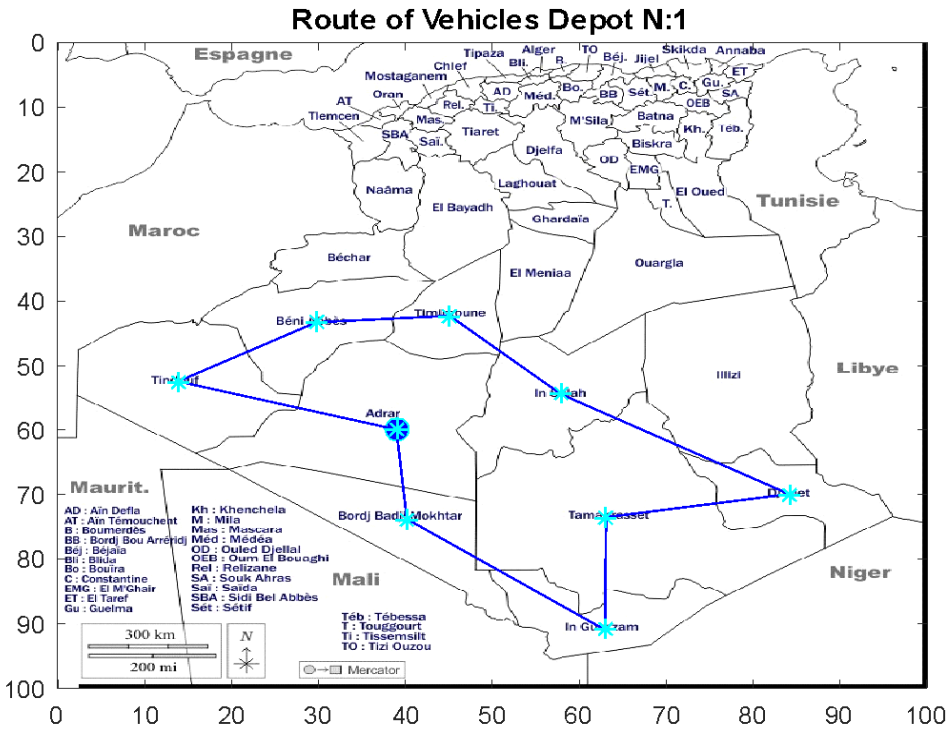

figure 5.11: the best solution obtained from the execution's series of capacity ,capacity>=700 ,Depot 1 paths

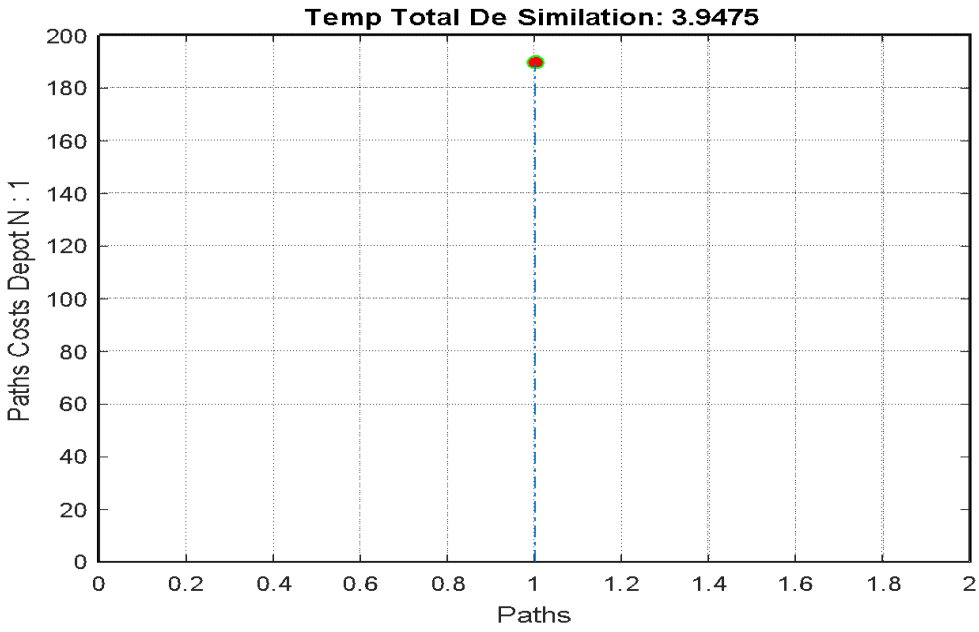

Figure 5.12: the best solution obtained from the execution's series of capacity >=700 ,depot 1 paths costs

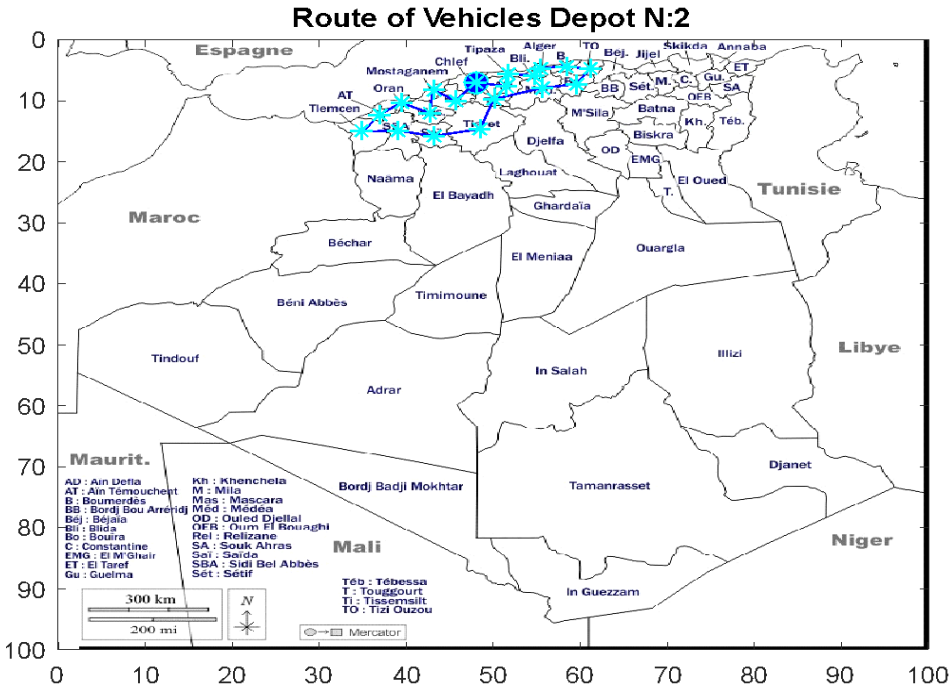

Figure 5.13: the best solution obtained from the execution's series of capacity>=700 ,depot 2 paths

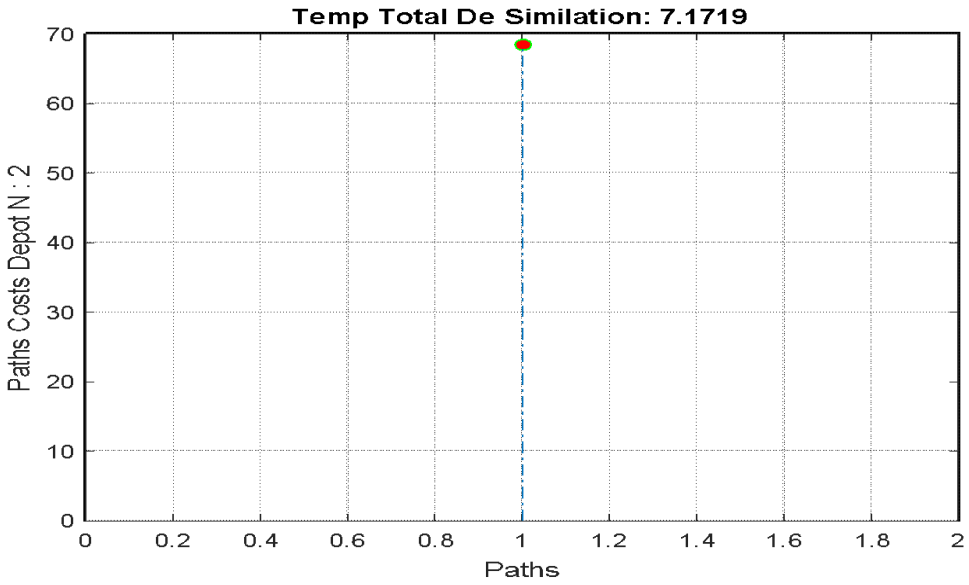

Figure 5.14:the best solution obtained from the execution's series of capacity paths costs

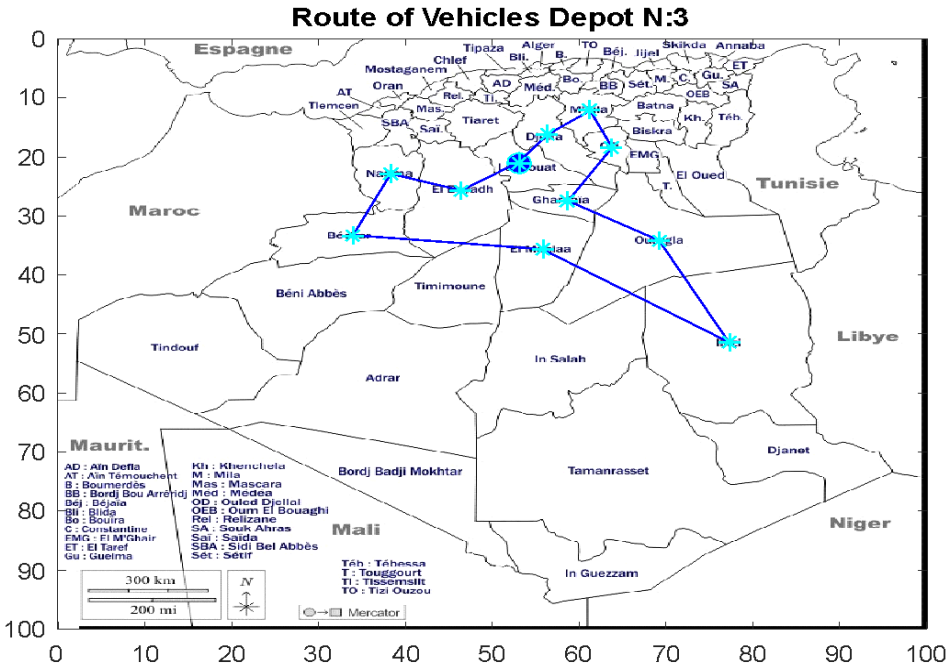

Figure 5.15: the best solution obtained from the execution's series of capacity >=700, depot 3 paths

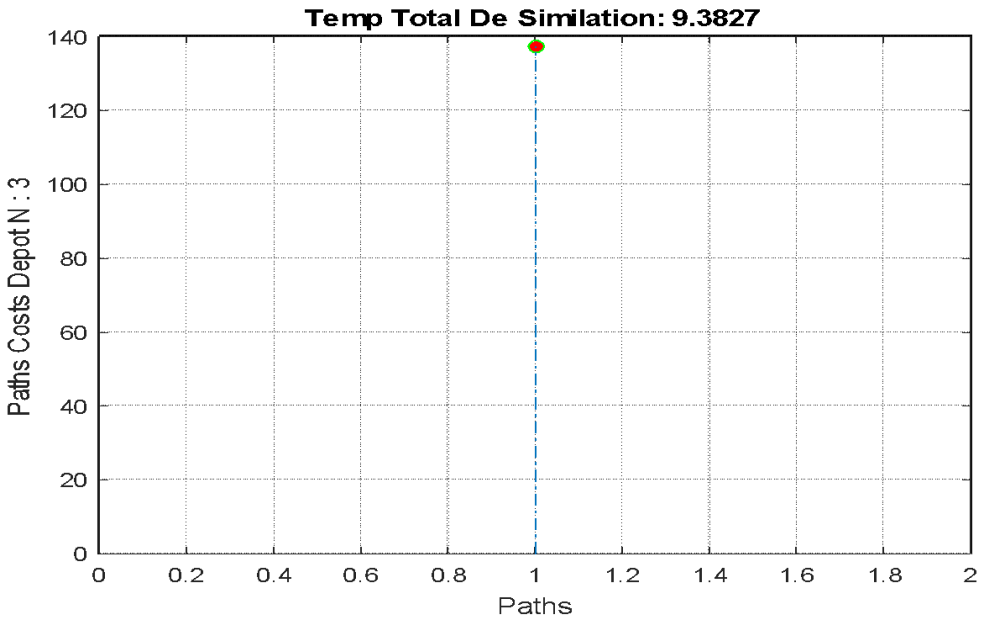

Figure 5.16: the best solution obtained from the execution's series of capacity

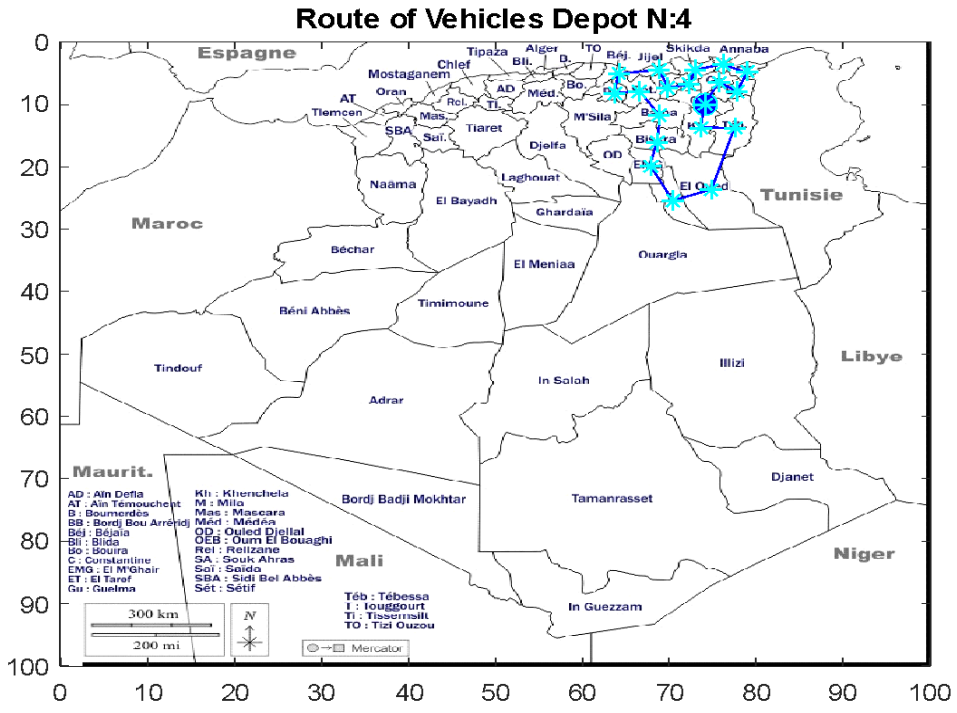

Figure 5.17: the best solution obtained from the execution's series of capacity>=700,depot 4 paths

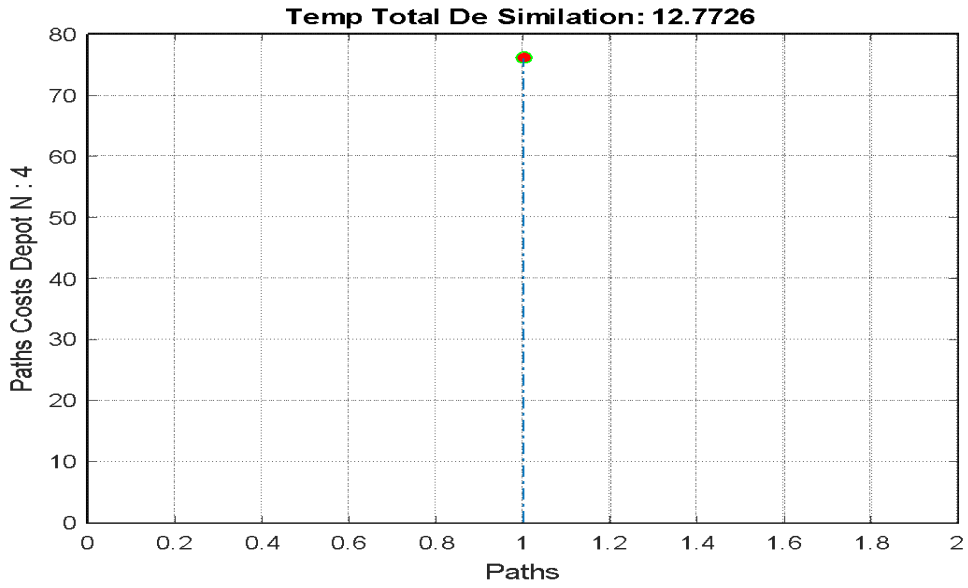

Figure 5.18: the best solution obtained from the execution's series of capacity>=700, depot 4 paths costs

5.4.7. The effect of iteration number on the results

We went to take 3 different number of iterations 100 200 300, and see what it gone happen to the results

5.4.1.a)1st first set of iteration number 100:

fix the values of the variables α = 1 , β = 1, maxIt = 50, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and put MaxIt=100.

table 5.20: statistics on the executions the 1st value of MaxIt=100

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	57.1758
2	1010.1565	57.6817
3	1010.1565	60.023
4	1010.1565	61.4096
5	1010.1565	60.3484
6	1010.1565	60.4094
7	1010.1565	67.8658
8	1010.1565	60.2213
9	1010.1565	61.1555

10	1010.1565	58.9335
Mean Sol	**1010.1565**	**60.5224**

5.4.1.b)2nd second set of iteration number=200:

We fix the values of the variables α = 1, β = 1, nAnt = 40, ρ = 0.65, capacity = 100, Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and put MaxIt = 200

table 5. 21: statistics on the executions of the 2nd value of MaxIt =200

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	114.9441
2	1010.1565	116.0685
3	1010.1565	115.544
4	1010.1565	115.9665
5	1010.1565	118.7432
6	1010.1565	116.8357
7	1010.1565	118.1859
8	1010.1565	118.3734
9	1010.1565	125.2581
10	1010.1565	115.3633
Mean Sol	**1010.1565**	117.5283

5.4.1.c)3rd third set value of α:

We fix the values of the variables = 1, β = 1, , nAnt = 40, ρ = 0.65, capacity = 100,

Q = 1, tau0=10*Q/(n_vertices*mean(model(idd).D(:))),ρ=0.65, and put MaxIt = 300

table 5.22: statistics on the executions of the 3rd value of MaxIt=300

Executions	Total cost of the overall solution	Total Time of Execution
1	1010.1565	172.0173
2	1010.1565	168.5768
3	1010.1565	157.943
4	1010.1565	155.9208
5	1010.1565	162.7767
6	1010.1565	154.3833
7	1010.1565	166.4022
8	1010.1565	165.3653
9	1010.1565	160.5645
10	1010.1565	164.5689
Mean Sol	**1010.1565**	**162.8519**

Comment: the results stay the same in the 3 series of experiment .

Result: the number of iteration MaxIt doesn't affect on the obtained solution from this model with this implementation , the solution is the same as is shown in the following screenshots :

5.5 conclusion

We conclude from the results that hybrid Ant Colony optimization algorithm can effectively deal with multi depot vehicle routing problem and 2opt algorithm can improve the time of search of the algorithm for the best solution tours and optimize it.

With the recent used model of 4 depot we see that the capacity of vehicle can improve the obtained solution by increasing the capacity of the vehicle we can solution with lower cost, but for the other parameters we don't see any difference in the total cost of the solution except for iteration number can reduce the time of simulation (execution) , more the number of the iteration is great more the execution time is longer , but it can make deference in other huge model in the cost of solution , the same thing for the other setting it can obtain more accurate solution .

General conclusion & Perspectives

In this report we have seen generalities in optimization, the methods of solving an optimization problem, and some of the most common problems in optimization.

and we explain the ant colony optimization algorithm, and we mention some of the optimization techniques, nearest distance clustering, and local interchange operation.

Hybrid Ant colony algorithms are one of the metaheuristic algorithms used to deal with vehicle routing problems and their variants, it's an algorithm inspired by the behaviour of ants forming a super organism and which constitutes a family of optimization metaheuristics.

This algorithm is based on the Ant System algorithm, which is originally created for solving the travelling salesman problem.

MDVRP has wide application scenarios, In order to adapt the ACO, we implement a hybrid ant colony optimization algorithm (HACO) to solve MDVRP. The HACO is based on ACO solving VRP and the nearest distance cluster for dealing with a multi depot, meanwhile 2-Opt to optimize routes

limits the implementation it doesn't obtain better cost after the 2-opt in the used models of MDVRP created by the implementation and it needs powerful hardware

finally, we test the implemented algorithm through multiple series of executions, and we try to determine the effect of changing the values of parameters of ACO, in order to obtain better solutions and better values of parameters.

In the future and as a perspective, it will be interesting to the Dynamic aspect of this work

because the VRP problem in the real is dynamic and variable and the services of delivery it obligate to considering the service hour and time constraints required by customers.

References

[1] Paper A Hybrid Ant Colony Optimization for Dynamic Multidepot Vehicle Routing Problem School of Computer Science and Technology ,Hangzhou

Dianzi University ,Hangzhou , China 2018

[2] BAYOU Abdelwahab BENSEFIA Amir A HYBRID ALGORITH (Aco-2opt) for solving the traveling salesman problem. El Bachir El Ibrahimi - BBA 2020/2021

[3] Bara Houria SUBJECT Development and implementation of a hybrid method for solving the quadratic assignment problem Universite Mohamed Boudiaf - M'sila 2019 .

[4] Combinatorial optimization and railway infrastructure capacity problemsXavier Delormeen 2003 à Valenciennes

[5] Catherine Mancel. Modelling and solving combinatorial optimization problems arising from space applications. Automatic / Robotic. Toulouse INSA, 2004

[6] (PDF) Vehicle routing problem: Models and solutions (researchgate.net), Date:04/06/2022 4:15

[7] Takwa Tlili, Sami Faiz, Saoussen," on Solving the multi depot vehicle routing problem", ReasearchGate,vol16.

[8] Haitao Xu,Pan Pu,and Feng Duan "A Hybrid Ant Colony Optimization for Dynamic Multi Depot Vehicle Routing Problem ,Hindawi research article,vol 18 ,pp2-5

[9] https://www.linkedin.com/pulse/vehicle-routing-problem-its-variants-sajaykumar-j/ Date: 11:40 06/04/2022

[10]Christian Blum Ant colony optimization: Introduction and recent trends 1 ALBCOM, LSI, Universitat Politècnica de Catalunya, 2005

[11] Marco Dorigo Thomas Stützle, Ant Colony Optimization Massachusetts Institute of Technology A Bradford Book The MIT Press Cambridge, Massachusetts London, England 2004

[13]12 Ant Colony Optimization: A Swarm Intelligence based Technique International Journal of Computer Applications (0975 – 8887) Volume 73– No.10, July 2013

[14]https://www.vocabulary.com/dictionary/ant#:~:text=An%20ant%20is%20an%20insect,divided%20into%20jobs%20or%20castes Date19:30 12/04/2022.

[15]https://www.researchgate.net/figure/Analogy-between-Natural-and-Artificial-Ants_tbl1_265161384date10:30 17/04/2022

[16]https://www.sciencedirect.com/topics/engineering/ant-colony-optimization, Date:06/06/22 11:52

[17]https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/qtc2.12023 Date:05/04/2022 10:15

[18]https://towardsdatascience.com/the-inspiration-of-an-ant-colony-optimization-f377568ea03f, consultation Date : 11/05/2022 13 :23

[19]https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms,Date:11/05/2022 17 :15 PM

[20] Rafsanjani, Zahra Asghari Varzaneh Marjan Kuchaki "Edge detection in digital images using Ant Colony Optimization'',Computer Science Journal of Moldova, vol.23, no.3(69), 2015

[21] R. van der Put "Routing in the faxfactory using mobile agents". Technical Report R&D-SV-98-276, KPN Research, 1998

[22] Dorigo Marco,Stützle, Thomas,Darmstadt, Tu,Group, IntellecticsThe Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, ResearchGate,March 2001

[23] HaitaoXu,PanPu,FengDuan "A Hybrid Ant Colony Optimization for Dynamic Multidepot Vehicle Routing Problem",Hindawi,pp3-9,Volume2018,id3624728

[24] https://www.techtarget.com/whatis/definition/MATLAB

[25] Kenza Bechtoula,Hadjer Bouguerra. Algorithme hybride (aco-ape) pour La Résolution du problème de Voyageur de commerce El Bachir El Ibrahimi 2020/2021

[26] https://www.pestworld.org/pest-guide/ants/ Date:11/05/2022 13 :15 PM

[27] Jianping Wang practicum submitted to the Faculty of Graduate Studies of The University of Manitoba 2008

Appendix A

implementation of Hybrid Ant Colony optimization for CMDVRP

A.1. Definition of the used map:

A.1.1 Definition of the used map:

```
function [Problem]=Algeria_Problem()
%Column 1 = customer number, no.1 is the deposit.
%Column 2 is the x-coordinate of the client
%Column 3 is the client's y-coordinate
%Column 4 is the demand of what the customer needs to supply, that of the deposit=0
% Column 5 is the duration time in hours of the service for the client
% Column 6 is the minimum time of hours of the initial dispatch for the client (time window)
% Column 7 is the maximum time of dispatch hours for the client (time window)
Problem=...
  [ 1      39.00      60.00       0.00      0.00      0.00      inf     %Adrar
    2      48.00      7.00       10.00      0.00      0.00      inf     %Chlef
    3      53.00      21.00      15.00      0.00      0.00      inf     %Laghouat
    4      74.00      10.00      7.00       0.00      0.00      inf     %Oum El Bouaghi
    5      68.90      11.90      30.00      0.00      0.00      inf     %Batna
    6      64.10      5.10       13.00      0.00      0.00      inf     %Bejia
    7      68.60      16.00      18.00      0.00      0.00      inf     %Biskra
    8      34.00      33.30      24.00      0.00      0.00      inf     %Bechar
    9      54.80      5.91       36.00      0.00      0.00      inf      %Blida
   10      59.50      7.37       6.00       0.00      0.00      inf      %Bouira
   11      63.10      73.60      10.00      0.00      0.00      inf     %Tamanrasset
   12      77.50      13.90      8.00       0.00      0.00      inf     %Tebessa
   13      35.00      15.10      5.00       0.00      0.00      inf     %Tlemcen
   14      48.50      14.60      16.00      0.00      0.00      inf    %Triaret
   15      61.10      4.84       29.00      0.00      0.00      inf      %Tizi Ouzou
   16      55.50      4.75       14.00      0.00      0.00      inf      %Alger
   17      56.40      16.20      10.00      0.00      0.00      inf     %Djelfa
```

Figure A.1: Definition of the map Algeria provinces

This script represents the definition of the Algeria model the first column 1 represents the indices

Column 2: represents the X coordinates, Column 3: represent the Y coordinates of provinces used the map. , the 4th Column represents the required quantity of goods for each customer .

A.2.Creation of the model used by H-ACO for CMDVRP:

```
function [model,Peter]=CreateModel(capacidad,numberdepots)
    %Numdepots=numberdepots;
    Problem=Algeria_Problem();
    Model_1=CreateModel_N();
    [depots]=clustering_algrotithms_based_Euclidian_distance4(Model_1,numberdepots);
   %------------------------Definition Struct of the model for ACO------------
    empty_model.z=[];
    empty_model.rutas=[];
    empty_model.n=[];
    empty_model.x=[];
    empty_model.y=[];
    empty_model.d=[];
    empty_model.D=[];
    empty_model.T=[];
    empty_model.municipio={};
    model=repmat(empty_model,numberdepots,1);
```

Figure A.2:part1 of creation model script

In this part we define the structure of the model ,which will give to aco algorithm .

```
 %read clients file
 for f=1:numberdepots
 x= depots(f).x_customers;%array in which it stores the coordinates in x of the clients
 y= depots(f).y_customers;%array in which it stores the coordinates in and of the clients
 Demanda= depots(f).demands;%array in which you store customer demands
 Peter=0; %this is boolean vcariable used to verify the condition of capacity %
 n=numel(x); %this variable is used to handle with nubmber of vertices of each submodel %
 D=depots(f).D; %%this variable is used to handle with Distance between vertices of each submodel %
 cliente=zeros(n);
 for i=1:n
    cliente(i)=i; % is used to have the indices of the ordered clients in a parallel array
 end
for i=1:n
model(f).cliente(i)=cliente(i);
model(f).municipio(i)=depots(f).willaya(i);
end
```

Figure A.3:part2 of creation model script

In this part we read the X and Y coordinates of depot f customers and their demand obtained from clustering function ,and we initialize some matrix to handle with the model .(see the comments in the figures for more detail)

```
model(f).clienteServido= zeros(n);%arrangement of clients with dispatch information
%which will have values of 1 and 0 (initialized at 0)
model(f).clienteServido(1)=1; % is omitted
cont=1; %1
ind=1;
rutas=0;% to know the number of routes generated
```

Figure A.4:part3 of creation model script

Part 3 creation model : define some variable to return it in with the model struct ,to know the served customers

```
while cont <= n %verifies that there is at least one client without serving min(model.clientServed)==0
     rutas=rutas+1;
     cont=cont+1;
        if cont <= n
         model(f).clienteservido(cont)=1;% means served
         disponibilidad= capacidad- Demanda(model(f).cliente(cont));
         %%disponibility =capacityofVehicle-somme(demandofcustomers of the model)
         z(ind).ruta=(model(f).municipio(cont));
         %[def<w] carries the record of the route, but the name of the municipalities
         ff(ind).ruta=(cliente(cont));%gets the path record, but only the indices
         while cont<numel(Demanda) && Demanda(model(f).cliente(cont+1)) <= disponibilidad
             if Demanda(model(f).cliente(cont+1)) <= disponibilidad
                 %the position that follows, also applies in the route
                cont=cont+1;
                z(ind).ruta=[z(ind).ruta model(f).municipio(cont)];
                ff(ind).ruta=[ff(ind).ruta,cliente(cont)];
                disponibilidad= disponibilidad - Demanda(model(f).cliente(cont));
                % the current capacity of the vehicle is updated
                model(f).clienteservido(cont)=1;% means served
             end
         end
         z(ind).ruta=[depots(f).willaya(1), z(ind).ruta];% the deposit is added to the route
         ff(ind).ruta=[1,ff(ind).ruta];%path of client indexes
         nn=numel(z(ind).ruta); %number of items in that path
         z(ind).nn=nn;%number of items in that path
```

```
         z(ind).ruta=[depots(f).willaya(1), z(ind).ruta];% the deposit is added to the route
         ff(ind).ruta=[1,ff(ind).ruta];%path of client indexes
         nn=numel(z(ind).ruta); %number of items in that path
         z(ind).nn=nn;%number of items in that path
         z(ind).ff=ff;
         ind=ind+1;
    end
end
```

Figure A.5:part4 of creation model script

This part of the script generates initial routes after checking the possibility of generating these routes according to the capacity and disponibility constraints of the vehicle, it also checks if each customer is served in these routes.

```
    rutas=rutas-1; % is placed in the corresponding value
    model(f).z=z; %has the routes and the cost matrices
    model(f).rutas=rutas; % this is empty in the end of this function will be implemented in ACO algorithm
    model(f).n=n;
    model(f).x=x;
    model(f).y=y;
    model(f).d=Demanda;
    model(f).D=D;
    end
end
```

Figure A.6:part5 of creation model script

In the end of each iteration for each submodel (depot + their customers) we return the results (initial routes) in defended struct model.

A.3.Nearest Distance Clustering function:

```
function  [depots]=clustering_algrotithms_based_Euclidian_distance4(model,numberdepots) %P1:model%P2:numberdepots
   % number_depot=input('Enter the number of depots you want it in the model to start with it');
PModel=Algeria_Problem();
model=CreateModel_N();   %
municipio= {'Adrar ','Chlef ','Laghouat ','Oum El Bouaghi ','Batna ','Bejia ','Biskra ','Bechar ','Blida ','Bouira
  % depot Matrix  %
    demands=PModel(:,4);
    %Empty depot %
    empty_depot.x=[];
    empty_depot.y=[];
    empty_depot.x_customers=[];
    empty_depot.y_customers=[];
    empty_depot.willaya={};
    empty_depot.demands=[];
    empty_depot.n=[];
    %depot struct %
    depots=repmat(empty_depot,numberdepots,1);
    %
    distancedepotcustomer=model.D(1:numberdepots,numberdepots+1:model.n);
    % we have choise explicitely the vertices of depot[1-numberdepots fisrt provinces] and vertices of customers
    %[numberdepots+1:model.n]the remaining vertices of pairs x,y]%
```

Figure A.7:part1 of Clustering function script

this part 1 of clustering algorithm define the struct of depot, which contain x and y coordinates matrixes of the customers and the depots it self in the first columns the name of clustered provinces in willaya matrix also their demands for each depot ,each depot with their results of clustering is in the struct depots represent a row in these struct .

Also, we define the distance matrix between the selected warehouses and customers for the collection process.

```
%classification and assignemet of customer %
M_min=min(distancedepotcustomer);
 % this loop assign the depot coordinates and add them to their customers coordinates %
for num=1:numberdepots
    depots(num).x=model.x(num);
    depots(num).y=model.y(num);
    depots(num).x_customers(1)= depots(num).x;
    depots(num).x_customers(1)= depots(num).x;
    depots(num).y_customers(1)= depots(num).y;
    depots(num).y_customers(1)= depots(num).y;
    depots(num).demands(1)=0; % the depot dont have demand of good
end
[L,C]=size(distancedepotcustomer);
```

Figure A.8:part2 of Clustering function script

Part 2 clustering function: first of all calculate the vector M_min wich contain the minimum column of each column of the matrix of distance between depots and customers to use it later ,also it initializes the depots struct , also it affect the depot to x and y customer coordinate to handle with it like a final customer to visit it.

```
   [L,C]=size(distancedepotcustomer);
     matrice_indice=ones(L); % this matrix help to the permutation of customer in the matix of depot
    for q=1:C
        for f=1:L   %checked
         if (distancedepotcustomer(f,q)==M_min(q))
             %this condition permet to cluster each customer to their nearest depot
         depots(f).x_customers(matrice_indice(f)+1)= model.x(q+numberdepots);
%Assigning customer x to repository depots, q + numberdepots is the pointer starting pointer for customers a
         depots(f).y_customers(matrice_indice(f)+1)= model.y(q+numberdepots);
%Assigning customer x to repository depots
         matrice_indice(f)=matrice_indice(f)+1; %Incremenation of matrix of indexes
         end
        end
    end
```

Figure A.9:part3 of Clustering function script

Part 3 clustering function : define matrice_indice matrix to permit each clustered customer to depots matrix ,also it do the clustering process by using the M_min vector wich contain the minimum value between each customer and the depots ,these values allow to get the nearest depot by searching in the matrix distance of (depot_customer) of the indice the this value which is the indice of the nearest depot (see it in the next part).

```
    for f=1:numberdepots
        indice_willaya=1;
      for i=1:numel(depots(f).x_customers)
          X=depots(f).x_customers(i);
          Y=depots(f).y_customers(i);
          %J=0;
          %I=0;
            bool=true;
          %islogical(J)==true && islogical(I)==true)
            while bool==true
          I=find(PModel(:,2)==X & PModel(:,3)==Y);
          depots(f).willaya(indice_willaya)=municipio(I) ;
          depots(f).demands(indice_willaya)=demands(I);
          indice_willaya = indice_willaya + 1;

      end
    end  %------------------------Fin Clustering Wilaya with demands -----------------%

  %End of funtion
```

Figure A.10:part4 of Clustering function script

Part 4 clustering function: in this part we get the names of provinces and their demand from the big model, by selecting the indices of the same X and Y coordinates of the clustered customers in the big model by find function.

In the end of this clustering function, we give the results to create model function which generate initial routes with these results.

A.3. HACO for MDVRP:

This implementation is modified from VRPTW script and adapted to be useful to the multi depot vehicle routing problem by clustering function , fusing with 2 opt function to get more optimized results ,2-opt can be effective for the big datasets in term of iteration and execution time .

```
%% Definition of the problem
tic; %start of calculating time of execution of the algorithm %
capacidad=100;  % capacity that can be loaded by the distributor vehicle
numberdepots=input('Entrer le nombre de depots a avoir'); % this variable for choise the number
%of depots to make it in the model %
[model,Peter]=CreateModel(capacidad,numberdepots);

for idd=1:numberdepots
    hold off
img = imread ('amp.jpg');%# Load a sample image
            min_x = 0;
            max_x = 100;
            min_y = 0;
            max_y = 100;
            figure(idd);
            x_min = min_x;
            x_max = max_x;
            y_min = min_y;
            y_max = max_y;
            imagesc ([x_min x_max ], [y_min y_max], img);
```

Figure A.11:H_ACO for MDVRP part1

```
            % ACO Parameters
            MaxIt=200;       % Maximum number of iterations
            nAnt=40;         %Number of Ants (Population Size)
            Q=1;             %control variable for pheromone generation
            alpha=1;        % follow pheromone intensity values
            beta=1;          % follow the values with respect to nearby cities (distances)
            rho=0.65;        % Pheromone Evaporation Rate

for r=1:model(idd).rutas%traverse all paths
    CostFunction=@(tour)TourLength(tour,model(idd),r);%CostFunction is a variable that will
    %contains the value returned by TourLength
    nVar1=model(idd).z(r).nn;%model.n;%numel(model.z(r).path) %nVar will
    %contain the total value of elements in the model array (x)
    nVar=model(idd).n;
    tau0=10*Q/(nVar*mean(model(idd).D(:)));% Initial pheromone stored in tau0
    % Initialization
    eta=1./model(idd).D; % Matrix of heuristic information
    tau=tau0*ones(nVar,nVar);    % Pheromone matrix
    BestCost=[];  %zeros(MaxIt*2,1); % Arrangement containing the best costs
     % empty ant
     empty_ant.Tour=[];
     empty_ant.Cost=[];
```

Figure A.12:H_ACO for MDVRP part2

```
 % Matrix of the ant colony
  ant=repmat(empty_ant,nAnt,1);
 % Ant candidate for best solution
   BestSol.Cost=inf;
% Main cycle of ACO
 if capacidad < max(model(idd).d) %verifies that the vehicle can meet the demands
     disp('The vehicle does not have the capacity to satisfy clients demand');
     break;
 else
     disp('ruta:');
     disp(r);
          cpt=MaxIt+1;
         for it=1:MaxIt
                 % Move Ants
             for k=1:nAnt
                 ant(k).Tour=model(idd).z(r).ff(r).ruta(1);
                 generated=(randi([2 nVar1])); %the ant chooses a random city
                 ant(k).Tour= [ant(k).Tour model(idd).z(r).ff(r).ruta(generated)];
                      for l=2:nVar1-1
                         i=ant(k).Tour(end); % i it contains the last node that was the ant visited
```

Figure A.13:H_ACO for MDVRP part3

```
             i=ant(k).Tour(end); % i it contains the last node that was the ant visited
             P=tau(i,:).^alpha.*eta(i,:).^beta; % compute the nominator of the probability
             %of the next node to visit it from the node
             %i (nominators of the probabilities  of selecting each node from i ) to choose
             %by kth ant
             P(ant(k).Tour)=0;  %the Initial Probability of the tour
             P=P/sum(P);  %compute the real probilities of travercing from each node to another node (Matrix
             %of problabilities).
             j=RouletteWheelSelection(P); % selecting the next node whith RoulleteWheelSekection
              AUX= zeros(size(model(idd).z(r).ff(r).ruta));
                 if ismember(model(idd).z(r).ff(r).ruta,j)==AUX %j does not belong to path
                        while ismember(model(idd).z(r).ff(r).ruta,j)==AUX
% while we can not go to the selected node ,is not accessible from the curent node do the follows steps
                         i=ant(k).Tour(end);
                         P=tau(i,:).^alpha.*eta(i,:).^beta;
                         P(ant(k).Tour)=0;
                         P=P/sum(P);
                         j=RouletteWheelSelection(P); %Selecting j with
                         %the function Routllete WheelSelection
                         end
                 end
          ant(k).Tour=[ant(k).Tour,j];
% The ant chose with Rolette whel selection a node to go in it (with the The process of Traping  the probability
%with the ascending accumulate probabilities)
```

Figure A.14:H_ACO for MDVRP part4

```
%with the ascending accumulate probabilities)
            end
    ant(k).Tour=[ant(k).Tour 1];
    ant(k).Cost=CostFunction(ant(k).Tour); %Compute the cost of tour that the k'th ant do it
    if ant(k).Cost<BestSol.Cost%checks if the cost of a certain ant is better than the best recorded
        BestSol=ant(k);
    end
end

% Update Pheromones
for k=1:nAnt

    tour=ant(k).Tour;

   % tour=[tour tour(1)]; %#ok add the first node visited  to the tour that is one of the constrait (rule)
   %in vehicle routing problems

    for l=1:nVar1

        i=tour(l);

        j=tour(l+1);

        tau(i,j)=tau(i,j)+Q/ant(k).Cost;  % aggregate the remaining pheromone
        %from evorating trails with the new deposited pheromone (new delta=Q(CONST)/legthofoverall(path);
```

Figure A.15:H_ACO for MDVRP part5

```
        %from evorating trails with the new deposited pheromone (new delta=Q(CONST)/legthofoverall(path);

    end

end


% Evaporation
tau=(1-rho)*tau; %Evaporation process at each link

Save the best cost
BestCost(it)=abs(BestSol.Cost);



% Show iteration information

disp([' Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

[BestSol.Tour,BestSol.Cost,Best]=exchange2(BestSol.Tour,model(idd).D,BestSol.Cost);  % 2-opt Algorithm process
    BestSol.Tour=tour;
    sizebes=numel(Best);
    for i=1:sizebes
    BestCost(end:end+i)=abs(Best(i));
    end
```

Figure A.16:H_ACO for MDVRP part6

A.4. 2exchange fonction (2-opt):

```
function [tour,Cost,Best] = exchange2(tour,D,Cost)
%EXCHANGE2 Improve tour p by 2-opt heuristics (pairwise exchange of edges).
%   The basic operation is to exchange the edge pair (ab,cd) with the pair
%   (ac,bd). The algoritm examines all possible edge pairs in the tour and
%   applies the best exchange. This procedure continues as long as the
%   tour length decreases. The resulting tour is called 2-optimal.
n = numel(tour);                % vertex number
Best=zeros([],1);               % Array to Hold Best Cost Values
Best(1,1)=Cost;
zmin = -Cost;
k=1;
% Iterate until the tour is 2-optimal
while zmin/Cost < -1e-6
    k=k+1;
    zmin = 0;
    i = 0;
    b = tour(n);                % Select the last vertex
    % Loop over all edge pairs (ab,cd)
    while i < n-2
        a = b;
        i = i+1;
        b = tour(i);            % select the second vertex
        Dab = D(a,b);           % Calculation of the Cost
        j = i+1;
        d = tour(j);            % select the third vertex
        % Calculation of the new Cost
```

Figure A.17:2-opt script part1

```
        % Calculation of the new Cost
        while j < n
            c = d;
            j = j+1;
            d = tour(j);        % select the forth vertex
            % Tour length diff z
            % Note: a == d will occur and give z = 0
            z = (D(a,c) - D(c,d)) + D(b,d) - Dab;
            % Keep best exchange
            if z < zmin
                zmin = z;
                imin = i;
                jmin = j;
            end
        end
    end
    % Apply exchange
    if zmin < 0
        tour(imin:jmin-1) = tour(jmin-1:-1:imin);
        Cost = Cost + zmin;
        Best(k,1)=Cost;
    end
end
```

Figure A.18:2-opt script part2

After finding each best solution tour in the end of all iteration, we try to optimize this tour by 2 opt, which exchange all possible pairwise of edges until the tour be optimized, if the tour is not optimizable the function lets the same tour.