People's Democratic Republic Of Algeria
Ministry Of Higher Education And Scientific Research
Hassiba Benbouali University Of Chlef

Faculty Of Exact Sciences And Computer Sciences
Departement Of Computer Sciences

Graduation Project For Obtaining The Licence Degree

Submitted by:
RAKAD Maria and SBAIHIA Nassim

Multimodal Transport Itinerary Optimization.

Filiere: Informatique.
Specialite: System d'information.
Option: Information System.

Defended on (10/07/2019) in accordance with decision of following committee members:

Belalia Aicha	Supervised by:
Meskine Fatima	Tahraoui Mohammed Amine

College Year: 2018/2019

Acknowledgements

We are deeply indebted to our principal supervisor Mr. *TAHRAOUI Mohammed Amin* who provided stimulating advice, guidance and encouragement to us every step of the way, for his constructive criticism and helpful comments, which helped us considerably in shaping up the work in a much better way than we had originally planned.

My special thanks to someone who inspired me when I was at the beginning of my road, to study for pursue a doctorate, I would also like to thank Mr.TAHAR ABBAS who was credited me with achieving what I aspired to.

We heartily thank our professors who have taught us more than we could ever give them credit for here. Also, we are grateful to all of those with whom we have had the pleasure to work during this and other related projects.

Last but not least, our deep and sincere gratitude to our families for their continuous and unparalleled love, help and support. we are grateful to our twins for always being there for us as a friend. we are forever indebted to our parents for giving us the opportunities and experiences that have made us who we are. They selflessly encouraged us to explore new directions in life and seek our own destiny. This journey would not have been possible if not for them, and we dedicate this milestone to them.

Abstract

The itinerary planning problem in public multimodal transport networks constitutes a common routing and scheduling decision faced by travelers. The objective of this thesis is to present a new algorithm for solving the itinerary planning problem, i.e., determination of the itinerary that optimizes a set of criteria (i.e. total travel time, travel distance, price and taken transport modes) while departing from the origin and arriving at the destination. The itinerary planning problem is expressed as a shortest path problem in a time schedule multimodal transport network. The proposed algorithm to solve the multi-modal itinerary problem is based on metaheuristic of *Ant Colony Optimization* (ACO). Experimental results have been assessed by solving real life itinerary problem defined on the transport network of the french city of Lyon. Results indicate that test problems were solved within reasonable amount of time and the new approach is efficient enough to be integrated within a real world itinerary planning system.

Contents

List of Figures

Introduction

When using multimodal transport, the transportation network is composed of multiple modes (e.g., bus, tramway, metro..). A common decision problem faced by travelers is choosing the optimal itinerary (a planned route or journey). Therefore, building Advanced Traveler Information Systems that provide passengers with pre-trip information on navigating through the network has become a certain need. Since passengers do not only seek a short-time travel, but they attempt to optimize other criteria such as cost, distance, used transport modes and effort (alternate between transport modes). an efficient routing system should incorporate a multiobjective analysis for both routes and transport modes.

The itinerary problem in multimodal transport networks is used to be represented as a multi-criteria (multi-objective) shortest path problem which is aiming at providing feasible route under certain needs. In single criterion era (the shortest path problem), Dijkstra algorithm has been considered as a representative solution. It is an exact algorithm which always determines the exact shortest one. However, real-world optimization problems can hardly be expressed with just one criterion, one result neither. When considering multi-criteria, conflict appears often, The exact algorithms could not handle it well. Therefore, in this study we propose a solution to the itinerary problem based on the metaheuristic of Ant Colony Optimization. The proposed algorithm solves the multi-criteria shortest path problem, as a study case criteria used ate *time* and *distance*.

The rest of this report is organized around four chapters as follows:

- **Chapter 1**: *Introductory Concepts*, In this chapter we introduce graph theory and its applications in real world, then some basic definitions and notations related to graphs. we also mention real networks representation and multimodal transport.

- **Chapter 2**: *Problem definition*, In this chapter, as the name mentions, we define shortest path problems and types, then we talk about available solutions. After we presented the problem we worked on.

- **Chapter 3**: *ACO-MCSP solution*, This chapter is dedicated to the proposed solution and the approach it follows. We begin by introducing the ACO (Ant Colony Optimization) metaheuristic, then we present the proposed solution and its sub-problems in details.

- **Chapter 4**: *Results*, In this chapter, test results of the application of proposed solution on several graphs, real networks and pseudo-randmoly generated are presented.

Chapter 1

Introductory Concepts

Graph theory is an important area in mathematics, it's the study of points and lines. In particular, it involves the ways in which sets of points, called vertices, can be connected by lines or arcs, called edges.

The basic idea of graphs was introduced first in 1735(18th century) by the great Swiss mathematician Leonhard Euler. He used graphs to solve the famous Knigsberg bridge problem. Figure 1.1 show the seven bridges of knigsberg

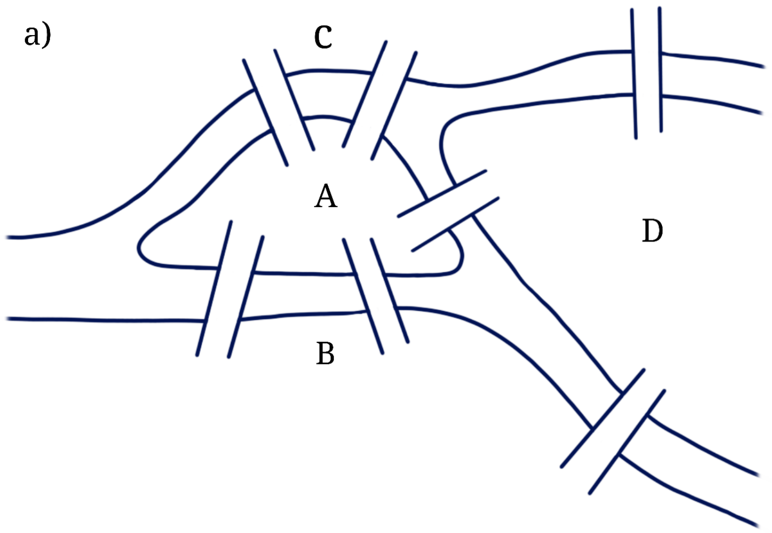


Figure 1.1: the seven bridges of knigsberg

German city of Knigsberg was situated on the river Pregel. It had a park situated on the banks of the river and two islands. Mainland and islands were joined by seven bridges. A problem was whether it was possible to take a walk through the town in such a way as to cross over every bridge only once.

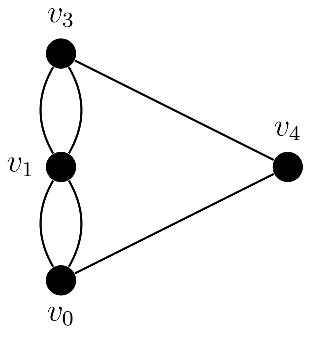


Figure 1.2: Graph of the seven bridges of knigsberg problem

As used in graph theory, the term graph does not refer to data charts, such as line graphs or bar graphs. Instead, it refers to a set of vertices (that is, points or nodes) and of edges (or lines) that connect the vertices.

1.1 Graph theory and it's applications

- In computer science, Graph theory can be introduced in many areas(workflow ,neural networks, google maps, fingerprint...).For instance ,the web page can be represented by a direct graph .The web pages are the vertices of graph and a directed edge is the link between two or more pages.
- Social networks represent a set of people or groups of people with certain interaction characteristics between them. For example, Facebook, LinkedIn, scientific collaboration network, business ties in a particular industry and labor markets. It permeates our social and economic lives.
- Networks are found in biological systems from food webs in ecology to various biochemical nets in molecular biology.For example, Protein interactions are another popular research area in cellular network. It can be considered as nodes and the links represent the physical interactions or binding.
- In the modern world, network not only refer to internet connection, it also refers to routes to move people, transport goods, communicate information and

control the flow of matter and energy. For instances, roads, railways, cables and pipelines are frequently represented and analyzed as a network.

- In mathematics, graphs are useful in matching. For instance , graph matching has applications in neural networks in artificial intelligence. For example, ANNs can perform image recognition on hand drawn digits.

1.2 Graphs

Graphs make it possible to reason about the relations between objects and how individual connections between objects can give rise to larger structures. For example, studying graphs makes it possible to reason about the overall distances between two cities, given their pairwise distances, or to find an optimal allocation of workers to tasks, given information on each individual worker's preferences.

A graph is a collection of objects (usually referred to as *nodes* or *vertices* that can be connected to one another. Specifically, any pair of nodes may be connected by an *edge* (sometimes called an *arc*).

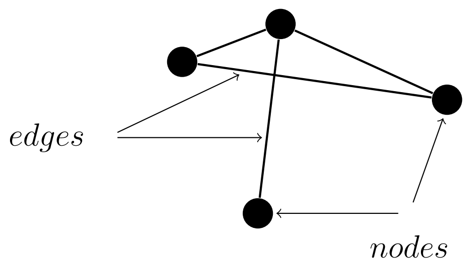


Figure 1.3: A graph with 4 nodes and 4 edges

1.2.1 Definitions

Now that we have an idea about graphs, nodes and edges, and how a set of nodes and edges constructs a graph we can go to some formal definitions.

Graph: A graph G is an ordered pair $G = (V, E)$, where V is a set of v*ertices* (or *nodes*) and E is a set of *edges* (or *links*), eache edge is formed by a pair of *vertices*, *vertices* can be labled with letters (e.g: a, b, c..) or numbers (e.g: *1*, *2*..) and edges can also hold data for example it may be distances in a roads network graph. The goal of a graph is representing the structure, so the same graph may have many apparences, Figure 1.4 shows the same graph in Figure 1.3 but labled in many apparences.

In this case, we could represent the above graph as follows:

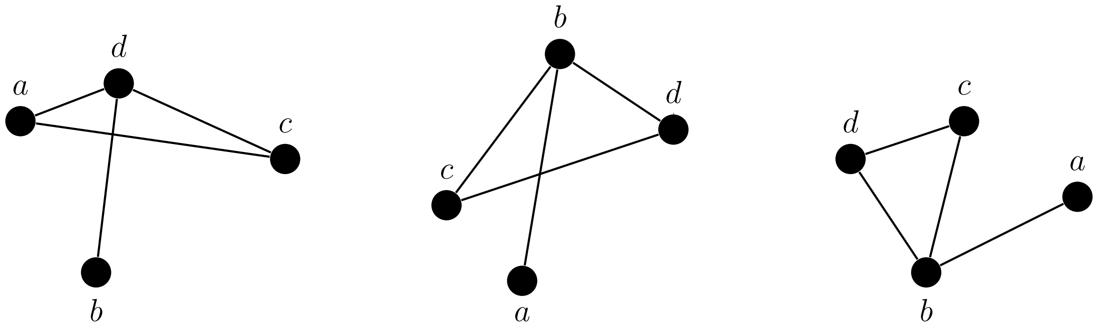


Figure 1.4: Same graph in 1.3 labled and in many apparences

$$G = (V, E)$$
$$V = \{A, B, C, D\}$$
$$E = \{\{A, B\}, \{B, C\}, \{B, D\}, \{D, C\}\}$$

Directed, undirected and oriented graph: An *undirected* graph is a graph $G = (V, E)$, where E is a set of *unordered* pairs. A directed graph (or *digraph*) is a graph where E is a set of *ordered* pairs, a directed graph is said *oriented* if it is having no symmetric pair of directed edges (i.e., edges may not be bidirected and point in both directions at once). The term graph without qualification often refers to *undirected* graphs.

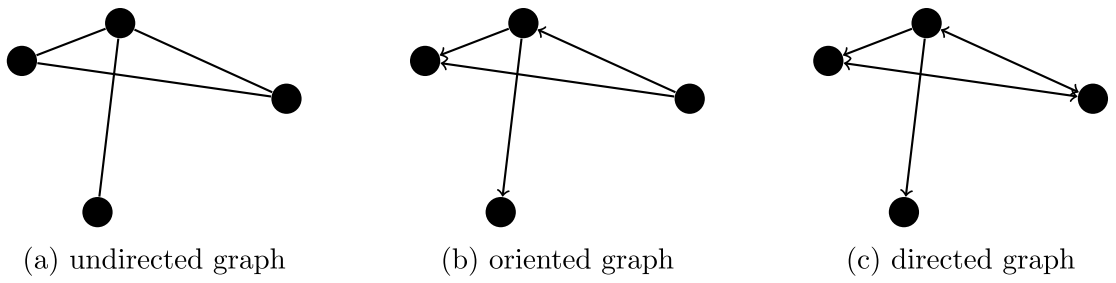


Figure 1.5: Directed, undirected and an oriented graph

Simple graph: A *simplegraph*, also called a *strict* graph, is graph containing no graph loops or multiple edges and it contains only one type of edges, all the previous examples are *simple* graphs. Graphs in Figures 1.3, 1.5 and 1.6 are exampels of simple graphs

Strongly connected graph: A graph is called *strongly connected* if there is a path in each direction between each pair of *vertices* of the graph. That is, a path exists from the first *vertex* in the pair to the second, and another path exists from the second *vertex* to the first. In a directed graph G that may not itself be strongly connected

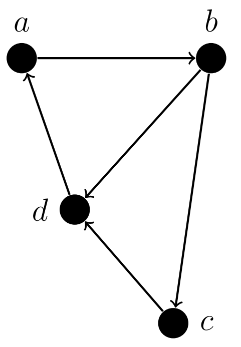


Figure 1.6: Example of a strongly connected di-graph

Multigraph: The term *multigraph* refers to a graph in which nodes can be connected to by multiple edges (more than one edge) from the same node, it may also have more than a one type of edges, the next figure shows an example of a *multigraph*.

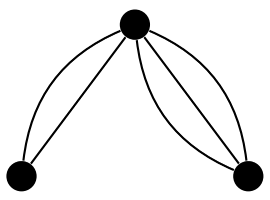


Figure 1.7: Example of a multigraph

1.2.2 Walks, trails and paths

Walk: A *walk* is defined as a sequence of alternating vertices and edges such as $v_0, e_1, v_1, e_2, ..., e_k, v_k$ where each edge $e_i = \{v_{i-1}, v_i\}$. The Length of this *walk* is k and if $v_0 = v_k$ we say it is a **closed walk**. For example, the graph below outlines a possibly *walk*, The walk is denoted as *abcdb*. Note that walks can have repeated edges. For example, if we had the walk *abcdcbce*, then that would be perfectly fine even though some edges are repeated.

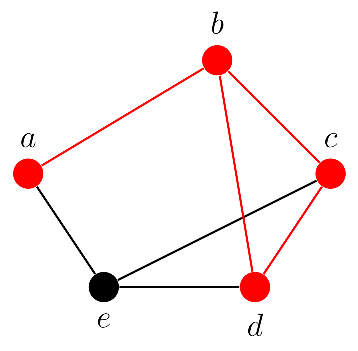


Figure 1.8: Example of a possible walk in a graph

Trail: A Trail is defined as a *walk* with no repeated edges. For example, the earlier example 1.8 can be considered a trail because there are no repeated edges.

Path: A path is defined as a walk with no repeated vertices. Notice that all paths must therefore be open walks, as a path cannot both start and terminate at the same vertex. For example, the following red coloured walk is a path from a to e or e to a since the graph is undirected.

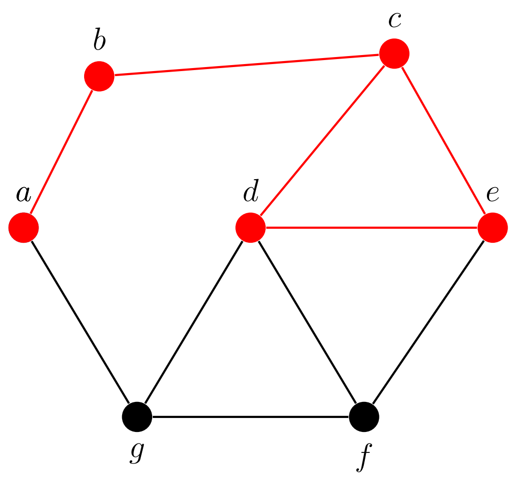


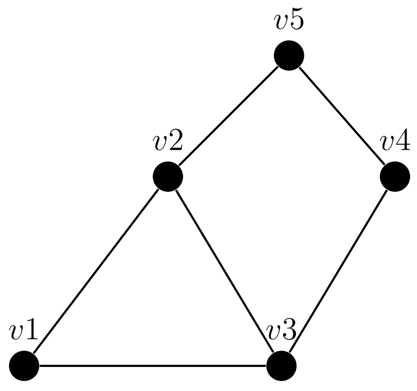
Figure 1.9: Example of a path in a graph

1.2.3 Graph representations

There are several ways to represent graphs; using an adjacency matrix, adjacency list, incidence matrix, incidence list and many others. The choice of the graph representation depends on the type of operations to be performed and ease of use.

Adjacency Matrix

Adjacency Matrix is a 2D array of size $V \times V$ where V is the number of *vertices* in a graph. A slot $adjM_{i,j} = 1$ indicates that there is an edge from vertex i to vertex j, and when $adjM_{i,j} = 0$ it means that there's no edge between the two vertices. Adjacency matrix for undirected graph is always symmetric.
Adjacency Matrix is also used to represent weighted graphs, if $adjM_{i,j} = w$ then there is an edge from vertex i to vertex j with weight w.



(a) Graph

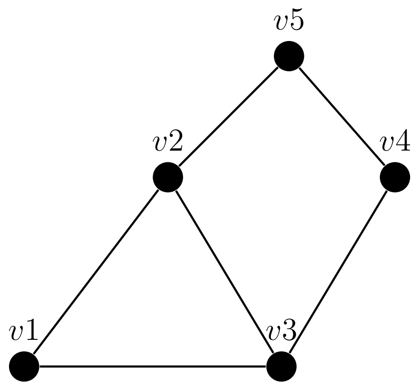
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

(b) Adjacency matrix

Figure 1.10: Example of Adjacency Matrix representation

Adjacency List

For each vertex we store an array of the vertices adjacent to it. We typically have an array of $|V|$ adjacency lists, one adjacency list per vertex. It is most useful for directed graphs. Here's an adjacency-list representation of the graph next to it:



(a) Graph

$$\boxed{v_1} \rightarrow \boxed{v_2}\boxed{v_3}$$
$$\boxed{v_2} \rightarrow \boxed{v_1}\boxed{v_3}\boxed{v_5}$$
$$\boxed{v_3} \rightarrow \boxed{v_1}\boxed{v_2}\boxed{v_4}$$
$$\boxed{v_4} \rightarrow \boxed{v_3}\boxed{v_4}$$
$$\boxed{v_5} \rightarrow \boxed{v_2}\boxed{v_4}$$

(b) Adjacency list

Figure 1.11: Example of Adjacency List representation of same graph in 1.10

1.2.4 Graph representation of real networks

Since the goal of a graph is representing the structure not the appearance of a network, a graph can be represented in many ways depending on the ease of use, as presented [4] the process follows some basic rules:

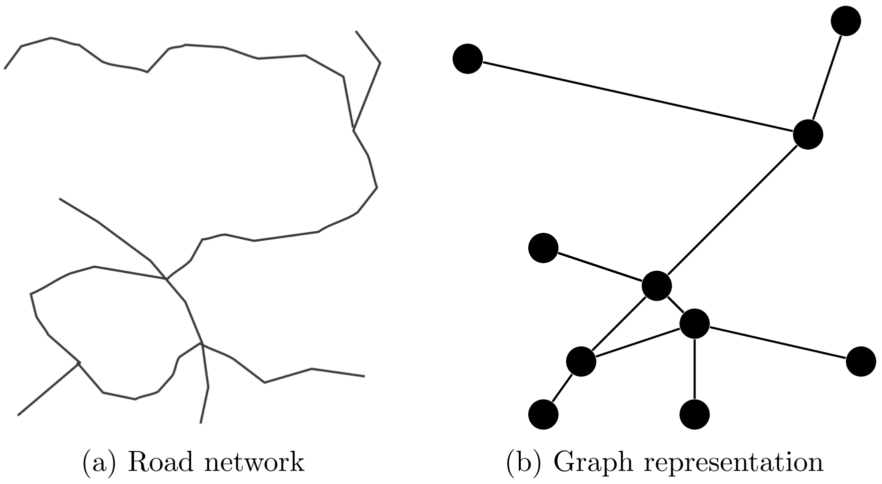

(a) Road network (b) Graph representation

Figure 1.12: Graph representation of a real road network

- The most important rule is that **every terminal and intersection point becomes a node**.
- Each connected node is then linked by a **straight segment**.
- A node that is not a terminal or an intersection point can be added to the graph if along that segment an **attribute is changing**. For instance, it would be recommended to represent as a node the shift from two lanes to four lanes along a continuous road segment, even if that shift does not occur at an intersection or terminal point.
- A **dummy node** can be added for aesthetical purposes, especially when it is required that the graph representation remains comparable to the real network.
- Although the relative location of each node can remain similar to their real world counterpart, this is not required.

Some transport networks such road, transit and rail networks which tend to be defined more by their links than by their nodes. This it is not necessarily the case for all transportation networks. For instance, maritime and air networks tend to be defined more by their nodes than by their links since links are often not clearly defined.

1.3 Multimodal transport problem

1.3.1 Definition

Multimodal transport is the combination of two or more means of transport to move passengers or goods from one source to a destination. During the displacement it can be used either public (e.g, bus, taxi, metro, railway, and ship). Multimodal transportation network can modeled as a graph whose sources and destinations are nodes and links are edges. (i, j) is an edge, it determins the *source* and *destination.*

1.3.2 Multimodal transport related definitions

Deprature *or* **Depart:** is the action of leaving place and moving to another, in multimodal transport a depart is characterized by a source and destination stations, a transport mode m and a time t. it is denoted as d_{ijmt}.

Itinerary: An itinerary is a schedule of events relating to planned travel, generally including destinations to be visited at specified times and means of transportation to move between those destinations. In an other way it is a sequence of departs $(d_1, d_2, d_3, ..., d_{k-1}, d_k)$.

1.3.3 Multimodal transport network graph representation

Let $G = (V, E, M)$ a graph for a transport network, where V is *node* set, E edge set and M the set of transport modes available in the network. Each station is considred as a *node*, and there is an edge between two distinct *stations* v_i, v_j if theres at least one depart d_{ijmt}, that means there is a depart from node v_i to v_j using the transport mode m at time t. the graph G is said to be *multimodal* because there are more than a way to move between nodes, it's also a *time dependent* graph because the cost of the depart d_{ijmt} at moment t_1 isn't the same as at t_2.

1.3.4 Problem of multimodal transport

One of the many problems in multimodal transport is finding itineraries (planned travel) based on the aim of travlers. A travler may favorate a specefied transport mode than an other, fastet itinerary (less traveling time) or a less distance itinerary. finding the optimal multi-criterias itinerary in multimodal transport networks is a shortest path problem but a multi-objective one, where objectives are aims of travler.

The next chapter introduce shortest path (SP) problems and defins the problem of itineraries in multimodal transport networks.

Chapter 2

Problem definition

Since finding itineraries in multimodal transport networks is a multi-objective (multi-criteria) shortest path problem **(MOSP** *or* **MCSP)**, in this chapter general (mono-objective) shortest path problem **SP** is introduced, then we define our main problem which is **MCSP on multimodal transport networks** and talk about its NP-completeness.

2.1 Mono-objective Shortest Path problem

Given a weight *graph* $G = (V, E)$, V is a set of vertices and E a set of edges, a source vertex s, and a destination vertex d, where $s, d \in V$, find the shortest-path between s and d that has the minimum weight (objective). The problem can be formulated as follows in (2.1), where c_{ij} is the cost of an edge (i, j) and x_{ij} is an indicator variable for whether edge (i, j) is part of the shortest path.

$$\text{minimize} \sum c_{ij} x_{ij} \tag{2.1}$$

The problem is also sometimes called the **single-pair shortest path** problem, to distinguish it from the exesting variations of the problem:

- **(SSSP) - Single-source shortest path problem**, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.
- **(SDSP) - Single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

- **(APSP) - All-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

2.1.1 SP Resolution methods

Mono-objective shortest path problem can be solved by classical methodes (methodes based on graph theory).

Dijkstras algorithm solves the single source shortest-path **(SSSP)** problem from a given vertex to all other vertices in a graph. Dijkstras algorithm is used over directed graphs with non-negative weights. The algorithm identifies two types of vertices: **(1)** Solved and **(2)** Unsolved vertices. It initially sets the source vertex as a solved vertex and checks all the other edges (through unsolved vertices) connected to the source vertex for shortest-paths to the destination. Once the algorithm identifies the shortest edge, it adds the corresponding vertex to the list of solved vertices. The algorithm iterates until all vertices are solved. Dijkstras algorithm achieves a time complexity of $O(n2)$. One advantage of the algorithm is that it does not need to investigate all edges. This is particularly useful when the weights on some of the edges are expensive. The disadvantage is that the algorithm deals only with non-negative weighted edges.

Bellman Ford algorithm also solves the **(SSSP)** problem, it is capable of handling negative weights unlike Dijkstras algorithm. It operates in a similar manner to Dijkstras, where it attempts to compute the shortest-path but instead of selecting the shortest distance neighbor edges with shortest distance, it selects all the neighbor edges. Then, it proceeds in $n - 1$ cycles in order to guarantee that all changes have been propagated through the graph.

2.2 MCSP on multimodal transport networks

2.2.1 Definition

The problem of multi-criterias (multiobjective) shortest path (MCSP *or* MOSP) in multimodal transport network is defined in the following way: given a *directed* graph $G = (V, E, M)$, consisting of a *node* set V, edge set E and M is the set of the transport modes considered. We are given a source node s, a target node t and a time t_0 that denots the moments to start searching for SP from the *source* node s (depart time) to t. A solution p is an sequence of departs (itinerary) $(d_1, d_2, d_3, ..., d_{k-1}d_k)$ where each depart d_{ijmt} consist of an *edge* (i, j), transport mode m and a depart time d_t. We denote the set of all possible solutions (itineraries *or* SPs) as P. Moreover, we are given a multiobjective cost function $c(p)$ to minimize (2.2), where $p \in P_{sp}$, c_{ijmt} is the cost of depart d_{ijmt} and x_{ijmt} is an indicator variable for whether the depart d_{ijmt} is part of the shortest path.

$$\text{minimize } c(p) = \sum c_{ijmt} x_{ijmt} \tag{2.2}$$

NP-completeness of the multi-objective shortest path is already proved; The **MCSP** is classified as an **NP-hard** by [7], *Chapter*6 of [3] named "The Multiobjective Shortest Path Problem Is NP-Hard, or Is It?" also studies and proves the NP-completeness of the MCSP problem.

2.2.2 MCSP resolution methods

When talking about $\mathcal{NP}$-hard problems, polynomial methods such the mentioned before fails unless **NP=P**. The problem of **MCSP** can be solved using $Metaheuristics$, a genetic algorithm is proposed in [9] to solve the multi-modal route planning (itinerary) problem. Other approaches to solve the problem exist, using approximate reasoning as fuzzy logic and artificial neural networks (ANN) [8].

In this study, we are proposing a solution to the problem based on metaheuristic of Ant Colony Optimization **ACO**. *Chapter* 3 introduces ACO and texplains our solution in details.

Chapter 3

ACO-MCSP solution

When considering multi-criteria optimization, exact algorithms could not handle it, in this chapter we introduce metaheuristic of ACO and present our ACO based solution to solve the multi-criteria shortest path problem.

3.1 The Ant Colony Optimization Metaheuristic - ACO

Ant colony optimization is a probabilistic technique, initially proposed by Marco Dorigo in 1992 in his PhD thesis [2] for solving combinatorial optimization problems. ACO uses artificial ants which is a decentralized (self-organized) multi-agent systemn in swarm intelligence methods, based on the behavior of real ants searching for food.

3.1.1 Ant colony optimization (ACO)

Ant colony optimization (ACO) methodology is based on the ants capability of finding the shortest path from their nest to a food source. An ant repeatedly hops from one location to another to ultimately reach the destination (food). Ants deposit an organic compound called *pheromone* while finding a path.; *pheromone* is characterized by evaporation, the density of the pheromone is an indicator of the utility of that destination to build better solutions. Figure 3.1 shows the behavior of ants to find shortest path food.

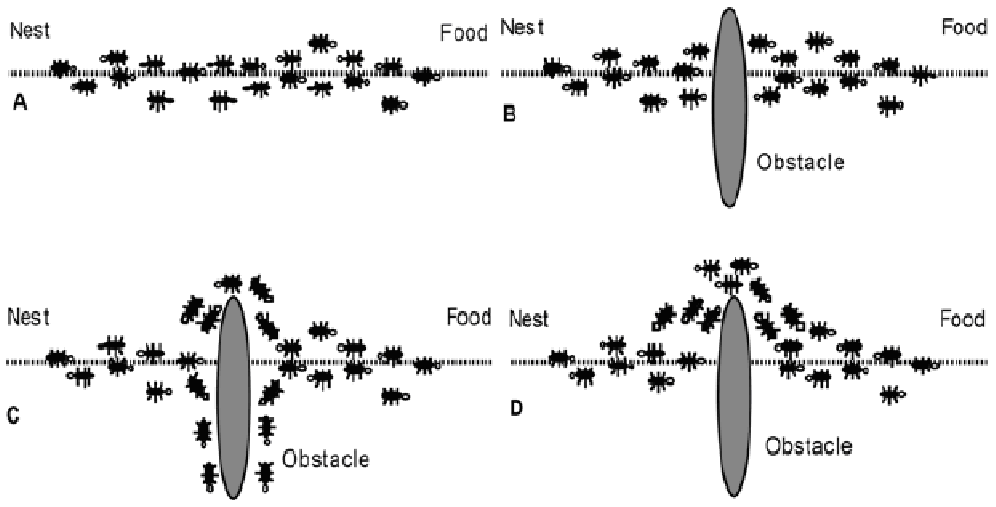


Figure 3.1: Real ants behavior to find paths to food

3.1.2 Ant colony optimization (ACO) methodology

At the start of ACO, ants are set on a source node, ants star moving between nodes searching for the targeted node (*destination* or *food*) based on probabilities which are dependent on the deposited pheromones. While moving, ants lay pheromones, after all ants construct solutions (found paths) pheromone trails are evaporated. Ants keep constructing solutions and laying pheromones, ants that find a shorter path are capable of doing more iterations (going frome the nest to food location) than other ants, in this way more pheromone is layed on shorter found paths. After a number of iterations ants start following the same path which is the the best paht found. Algorithm below shows how ACO metaheuristic works

Algorithm 1: procedure ACO methaheuristic

```
1 Set parameters, initialize Pheromone trail;
2 while stopping criterion not satisfied do
3     generateSolutions();
4     daemonAction();
5     pheromoneUpdate();
6 end
```

Two important steps in ACO metaheuristic are **edge selection** and **pheromone update**.

Edge selection (next node)

Ants chose nodes to move to by probabilities. For ant ant k, the probability to move from a node v_1 to v_2 passing by edge (i, j) is p_{ij}^k, Formula (3.1) shows how probabilities are calculated [2] [5], where τ_{xy} is the amount of pheromone deposited on egde (i, j), α is a parameter to control the influence of τ_{ij}, η is a heuristic information usualy $\eta = 1/C_{ij}$ where C_{ij} is the cost of edge (i, j), and β is a parameter to control the influence of the heuristic information η.

$$p_{ij}^k = \frac{(\tau_{ijmt})^\alpha (\eta_{ij})^\beta}{\sum_{(i,j)\in E}(\tau_{ijmt})^\alpha (\eta_{ij}^\beta)} \tag{3.1}$$

Pheromone update

When all the ants have completed a solution, pheromone trails are updated, as montioned in Formula (3.2, where τ_{xy} is the amount of pheromone deposited on egde (i, j), ρ is the pheromone evaporation coefficient and $\Delta\tau_{ij}^k$ is the amount of pheromone deposited by ant k. $\Delta\tau_{ij}^k$ equals 0 if edge (i, j) does not belongs to ank k solution.

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3.2}$$

3.1.3 ACO extentions

ACO has appeared in many variations (extentions), it differs in the way in which pheromone is updated.

- **Ant System (AS)** was the first ACO algorithm to be proposed in the literature [2]. Its main characteristic is that the pheromone values are updated by all the ants that have completed the tour.
- $\mathcal{MAX}$ -$\mathcal{MIN}$ **Ant System (MMAS)** is an improvement over the original Ant System idea, introduces the following two changes, only the best ant can update the pheromone trails, and the minimum and maximum values of the pheromone are limited.
- **Ant Colony System (ACS)** is another improvement over the original Ant System, the most interesting contribution of ACS is the introduction of a *local pheromone update* in addition to the pheromone update performed at the end of the construction process (called here $offline$ pheromone update).

- **Rank-based ant system (ASrank)** All solutions are ranked according to their length. The amount of *pheromone* deposited is then weighted for each solution, such that solutions with shorter paths deposit more *pheromone* than the solutions with longer paths.

3.2 Our Solution (ACO-MCSP)

Our solution (called **ACO-MCSP**) to solve the problem of multi-criteria itinerary in multimodal transport is based on the metaheuristic of ant colony optimization **ACO**. it is important to note that metaheuristic of ant colony has been developed for solving NP-hard problems, so the results obtained following its application provide approximations of optimum results.

The proposed ACO based algorithm, provides techniques to find approximations of best solutions to the problem of multi-criterias itineraries in a multimodal transport network, the creterias used in this study case are *time* and *distance*. Figure 3.2 is a flowchart of the algorithm, it explains the steps in which the algorithm operates.

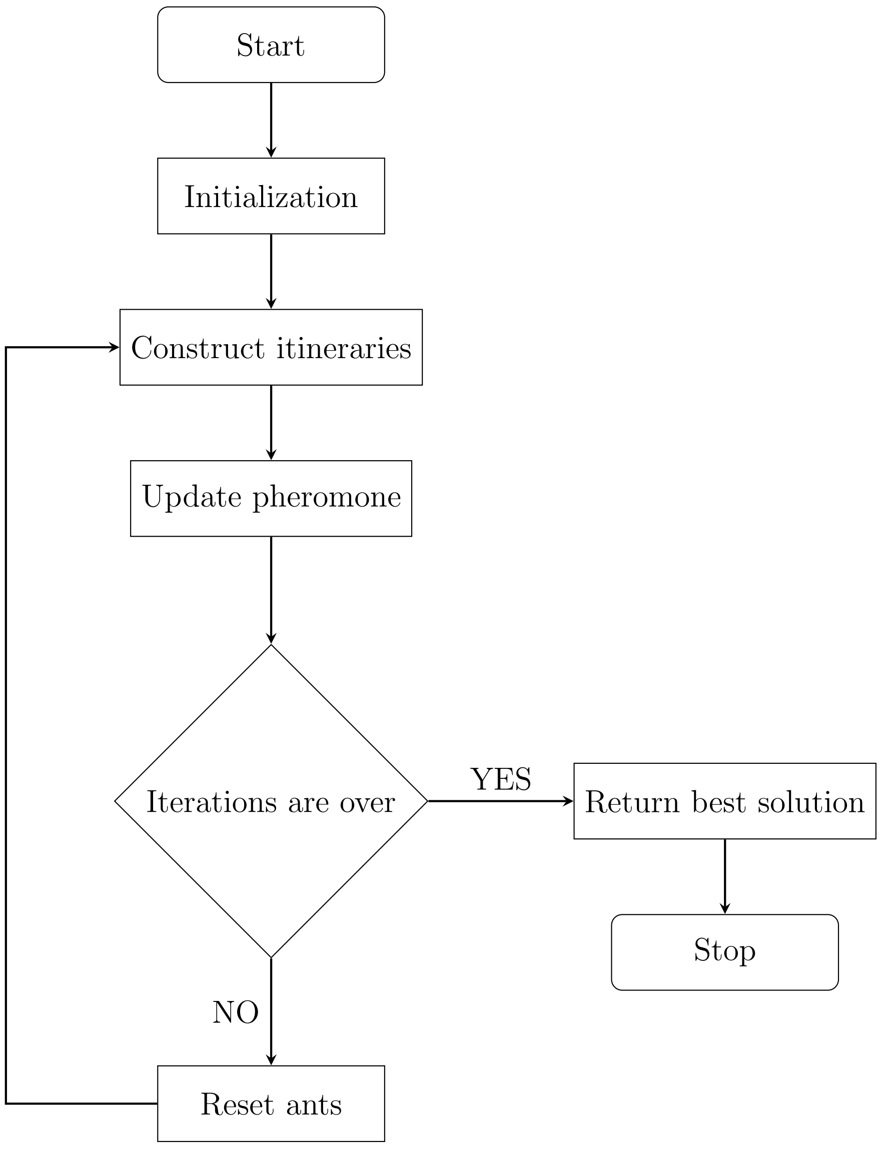


Figure 3.2: Flowchart of the proposed ACO based algorithm.

3.2.1 Solution overview

Our ACO-MCSP solution to the problem uses the following parameters:

- m number of ants
- i number of iterations
- t_0 depart time
- s source vertex
- d destination or targeted vertex
- α influence of pheromone on the choice of the next vertex
- β influence of the other criterias on the choice of the next vertex
- ρ parameter that defines the evaporation of the pheromone
- f_d,f_t parameters to define the importance of *distance* and *time* criterias
- τ_0 initial level of pheromone
- τ_{min},τ_{max} minimum and maximum level of pheromone

Figure 3.3: Parameters used in proposed ACO based solution

The number of ants m influences considerably the accuracy of a solultion, at the same time it increases the time of its operation. The α and β parameters allow us to modify the methode of the selection of the next vertex, which in turn influences considerably the quality of the solution. The parameter ρ decides the speed at which the pheromone trail evaporates, the quicker it is evaporated, the more possibilities arise to increase the exploration of ants to finds new solutions, whereas the slower evaporation makes ant use solutions that have been already found.
The parameters f_d and f_t are used to set the importance of criterias their values are between [0-1], more than a criteria can be considred to construct solutions (in our case *distance*,*time*).

At the start of the algorithm, and after setting the parameters of ACO; m ants are placed on the *source* node, and initialized given the needed informations to start searching for paths (itineraries) by probabilities and depose *pheromone* on each selected depart. The deposited *pheromone* evaporate by time, and ants keep doing the same proccess while the iterations are not over. After a while of time, ants start following the same path (taking the same departs to move between vertices) by pheromone which is the best discovred path. After the iterations are over, ants are leaded to a solution by the deposited pheromones. Algorithm 2 shows the steps of the algorithm.

Algorithm 2: ACO-MCSP **Input:** s, d, t **Output:** itinerary

```
setACOparameters();
initializePheromone();
while iterations are not over do
    foreach ant k ∈ M do
        while ant k is not at node t do
            calculateProbabilities(k);
            choseNextDepart(k);
        end
        updateBestSolution();
    end
    updatePheromone();
    evaporatePheromone();
end
return bestSolution;
```

Solution indepth

3.2.2 Algorithm subproblems in detail

Initialization

In the initialization phase, ACO parameters are setup, and the m ant are placed on the *source* node and reseted. In addition to that, the initial amount of pheromone is diposited on all departs, in our case the τ_0 is equal to τ_{min}.

Algorithm 3: initialize() **Input:** g,M$, s, d, t$

```
foreach vertex v ∈ V do
    $v_{visited}$ ← false;
    foreach edge e ∈ $E_v$ do
        foreach transport mode m ∈ $M_e$ do
            foreach depart d ∈ $D_e$ do
                $d_{pheromone}$ ← $\tau_{min}$;
            end
        end
    end
end
foreach ant m do
    $m_{currentVertex}$ ← s;
    $m_{currentTime}$ ← $t_0$;
    setVisited(m,s);
end
```

Finding paths

One of the important steps of the algorithms based on the use of the ant colony system paradigm are the rules for the process of establishing paths from the initial point to the end point. Many methodes can be used to establish paths, in the algorithm we are finding consecutively by each ant, the whole path in each iteration, and then updating the pheromone on these paths. Another methode is to move ants one by one to a subsequent vertex (taking an itenerary) and depose pheromone until they all find whole paths. In order to avoid favoriting paths with a lower number of edges, we are laying pheromone after all ants construct solutions; and the amount of the deposited pheromone depends on the ratio between the cost (criterias are considred) of a path and the cost of the best path which is found by an ant refered to by ant_{best}.

Another methode is presented in [6] using a time list. The time list is used for the purpose of recording times in which an ant reaches a given vertex, the ant that reaches the target vertex earlier will be able to lay down pheromones on the same path on which it arrived quicker. In this way it is possible to increase the pheromone trail on the edges of this path earlier than on the edges of longer paths.

Choice of next node

The choice of the next *depart* (taken *depart* defines the next *node*) is based on the calculated probabilities, the probability of ant k at node v chose the next depart to take is denoted by p^k_{ijmt} where (i, j) is the edge, m the transport mode and t is the time of the depart to take. As presented in [2] and [5], probabilities in general are calculated following the formula in (3.3), where Q is the coefficient of the depart.

$$p^k_{ijmt} = \frac{Q_{ijmt}}{\sum_{(i,j)\in E_v} Q_{ijmt}} \tag{3.3}$$

Calculation of depart coefficient considerably effects the solution, it controles the probabily to chose a depart, the used formula to calculate coefficient is shown beleow in (3.4) where τ_{ijmt} is the the current value of pheromone, d_{ijm} the distance of the transport line m and t_{ijmt} is the spent time to reach the next *node* (it is equivalent to the waiting time + travel duration). Before calculating probabilities, costs of itineraries are calculated for each ant.

$$Q_{ijmt} = \begin{cases} (\tau_{ijmt})^{\alpha}(\frac{1}{d_{ijm}f_d + t_{ijmt}f_t})^{\beta} + q & \text{if } v_{visited} = false \\ (\tau_{ijmt})^{\alpha}(\frac{1}{d_{ijm}f_d + t_{ijmt}f_t})^{\beta} & \text{else} \end{cases} \tag{3.4}$$

To increase ants exploration, an additional valaue q is added to not reached *vertices* (in the running iteration) by ants. Also to prevent faster convergence reaching some randomness is added by using a randomnes factor f_R.

algorithm below show the process of selecting next depart.

Algorithm 4: choice of the next vertex algorithm

```
1 $D \leftarrow$ getAllDepartsCosts($k$);
2 $P \leftarrow$ calculateProbabilities($D$);
3 if generatRandom() $\leq f_R$ then
4     visitRandomly($k$);
5 else
6     visitNextNode(P,D,k);
7 end
```

Updating pheromone

The way of updating the pheromone trail is related to the method for finding paths. Due to the fact that the pheromone trail deposited by ants influences decisions made by other ants to construct paths, pheromone is updated once every iteration and after all ants construct solutions. To positively affect the choice of ants and assure that ants search for a better solutions the amount of deposited pheromone is calculated dependent on the best found solution. Formula (3.5) shows how $\Delta\tau^k_{ijmt}$ the deposited pheromones by ant k on depart d_{ijmt} are calculated, where $C_{bestAnt}$ is the cost of the best found solution and C_{ant} is the cost of the solution of the ant that is going to lay pheromones.

$$\Delta\tau^k_{ijmt} = \begin{cases} \frac{C_{bestAnt}}{C_{ant}}\tau_{min} & \text{if } d_{ijmt} \text{ is taken by } ant\ k \\ 0 & \text{else} \end{cases} \tag{3.5}$$

code (5) below shows the process:

```
Procedure 5: updatePheromone(ant $k$)                    Input: ant $k$
foreach $d \in D^k_{visited}$ do
    $\tau_d \leftarrow \tau_d + (C_{bestAnt}/C_k)\tau_{min}$;
end
```

Evaporation of pheromone

The mechanism for evaporation of pheromones itself is easy and intuitive, the value of pheromones on depart is multiplied by the parameter ρ which cotrols the speed of evaporation. But because we are using the $\mathcal{MIN} - \mathcal{MAX}$ System a *minimum* and *maximum* values of pheromone are set. A partial exemption is applied on the departs that belongs to the solution found by the ant_{best}, formula (3.6) shows the process.

$$\tau = \begin{cases} \tau(\rho/2) & \text{if } d_{ijmt} \text{ belongs to } ant_{best} \\ \tau\rho & \text{else} \end{cases} \tag{3.6}$$

Algorithm below shows how the process of evaporating pheromones is done

```
Procedure 6: EvaporatPheromones()
foreach vertex v $\in V$ do
    foreach edge e $\in E_v$ do
        foreach transport mode $m \in M_e$ do
            foreach depart $d \in D_e$ do
                if isVisited($ant_{best}$, $d$) then
                    $\tau_{new} \leftarrow \tau_d(\rho/2)$;
                else
                    $\tau_{new} \leftarrow \tau_d\rho$;
                end
                if $\tau_{new} > \tau_{max}$ then
                    $\tau_{new} \leftarrow \tau_{max}$;
                if $\tau_{new} < \tau_{min}$ then
                    $\tau_{new} \leftarrow \tau_{min}$;
                $\tau_d \leftarrow \tau_{new}$;
            end
        end
    end
end
```

Chapter 4

Results

Experimental results are necessary to show the efficiency of our solution, we tested it on real transport networks (Lyon city transport network), and also on a pseudorandom generated multimodal networks, results are presented in this chapter.

4.1 Test environment

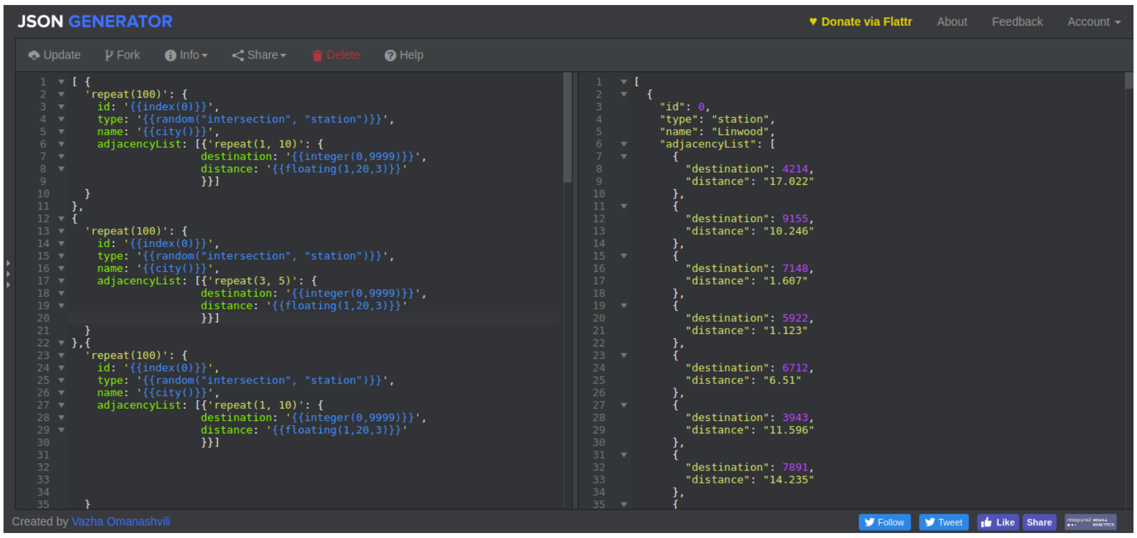
All explained algorithms are implimented in Javascript an run on the Node.js, a cross-platform JavaScript run-time environment. Node.js is based on googles fast v8 engine, The machine running the code consist of an intel i5-7th CPU and 8 GB of RAM.

4.2 Data sets

In order to test the ACO-MCSP, we needed datasets which are a multimodal transport networks graphs, because of lack of datasets we worked on algorithms to generate MTN graphs in a randmo and pseudo random way. We also cloned a real transport network of french city Lyon and transform it to a multimodal transport network using the algorithms we made. All datasets are in $JSON$ format which is easy to parse to objects in javascript.

4.2.1 Random generation

Graph generation consists of two phases, the first one is generating the network, and the second phase is filling the network with data such transport modes, departs and travel durations. In random generation, we used 2 methodes:



```
[ {
  'repeat(100)': {
    id: '{{index(0)}}',
    type: '{{random("intersection", "station")}}',
    name: '{{city()}}',
    adjacencyList: [{'repeat(1, 10)': {
                    destination: '{{integer(0,9999)}}',
                    distance: '{{floating(1,20,3)}}'
                    }}]
  }
},
{
  'repeat(100)': {
    id: '{{index(0)}}',
    type: '{{random("intersection", "station")}}',
    name: '{{city()}}',
    adjacencyList: [{'repeat(3, 5)': {
                    destination: '{{integer(0,9999)}}',
                    distance: '{{floating(1,20,3)}}'
                    }}]
  }
},{
  'repeat(100)': {
    id: '{{index(0)}}',
    type: '{{random("intersection", "station")}}',
    name: '{{city()}}',
    adjacencyList: [{'repeat(1, 10)': {
                    destination: '{{integer(0,9999)}}',
                    distance: '{{floating(1,20,3)}}'
                    }}]




  }
```

```
[
  {
    "id": 0,
    "type": "station",
    "name": "Linwood",
    "adjacencyList": [
      {
        "destination": 4214,
        "distance": "17.022"
      },
      {
        "destination": 9155,
        "distance": "10.246"
      },
      {
        "destination": 7148,
        "distance": "1.607"
      },
      {
        "destination": 5922,
        "distance": "1.123"
      },
      {
        "destination": 6712,
        "distance": "6.51"
      },
      {
        "destination": 3943,
        "distance": "11.596"
      },
      {
        "destination": 7891,
        "distance": "14.235"
      },
      {
```

Figure 4.1: JSON Generator Tool for generating random data

The first method is based on an online tool to generate Json Data called "JSON Generator Tool for generating random data" url(www.json-generator.com). Json generator takes as an input a code similar to json (the left part in Figure) but with a few functions and it outputs a Json Data (the riht part in Figure). Figures shows a screenshot of the tool.

The second method is an algorithm we made, the algorithm creats n layers of nodes and then start connecting each layer l_i with the next layer l_{i+1}, at the same time *transport data* such transport lines, distances and departs are generated and filled in the multimodal graph.

4.2.2 Pseudorandom generation

Pseudorandom generation generates errorless and more logical graphs that are similar to real MTN graphs, one problem of the first method in random generation is that it is possible to get a not strongly connected graph (contains more than one strongly connected component SCC); A problem of the second methode is that it's possible to find the same vehicle in depart d_{ijmt} in a deferent depart that is at the same time t.

In *Pseudorandom generation* this problem is solved by creating a set of transport lines, and assign a set of vehicles to each line and give it a set of departs then fill the transport data according to the vehicles.

4.2.3 Real graphs cloning

The lack of real multimodal transport data force us to clone real transport networks, the process of cloning multimodal transport networks is complex and gets more difficult the more data increases. To make the process of cloning easier, and avoid errors, we developed a small interactive command line user interface (CLI) using *Inquirer.js*. The developed CLI provide tests and guides the input, it also make it possible to continue from the last entred point.

4.3 ACO-MCSP parameters setting

ACO parameters effect considerably the result of the algorithm, appropriate setting of parameters requires both human experience and luck to some extend. From test results and others experiences and studies [1], a few hints can be considred when chosing parameters: Big values of parameters i can be useless since the result of the algorithm is related to the convergence. The number of ants m should be set according to the size of the graph. Other important parameters are those of pheromones, at most of time α should be greater than β, the deposited pheromone value Δ_τ is set according to the other costs (such *distance* and *time* in our case) to not lose the impact of pheromones on decesion making, quick evaporation of pheromones effects positively ants exploration and convergence. Tabel 4.1 shows some tests to set ACO parameters (i and m).

ACO parameters				Results (time, distance)		
i	m	s	d	good	bad	time (ms)
10	15	venissieux	charpennes	(115, 86.1)	(707, 81.2)	147.1
15	10	venissieux	charpennes	(115, 86.1)	(1067, 81.2)	141.6
20	15	venissieux	charpennes	(115, 86.1)	(1515, 86.1)	270.6
40	20	venissieux	charpennes	(87, 63.1)	(3009, 106.6)	656.4
40	35	venissieux	charpennes	(87, 63.1)	(3009, 106.6)	1061.7
50	50	venissieux	charpennes	(87, 63.1)	(4409, 167.1)	2154.8

Table 4.1: ACO-MCSP applied on MTNs

4.4 Tests and results

In the following tests, parameters shown in tabel 4.2 are used for ACO-MCSP. The number of iterations i, ants m, source node s and destination node d are changed according to test graphs.

m	i	t_0	s	d	α	β	ρ	f_d, f_t	τ_0	τ_{min}, τ_{max}
x	x	$08:88$	x	x	0.6	0.4	0.2	$(0.4, 0.6)$	τ_{min}	$(1, 6)$

Table 4.2: ACO-MCSP parameters setting

4.4.1 Real transport networks

TCL Lyon transport network

"The Transports en Commun Lyonnais" (TCL) is the public transport agency of french city Lyon. It is the second largest public transport system in France.

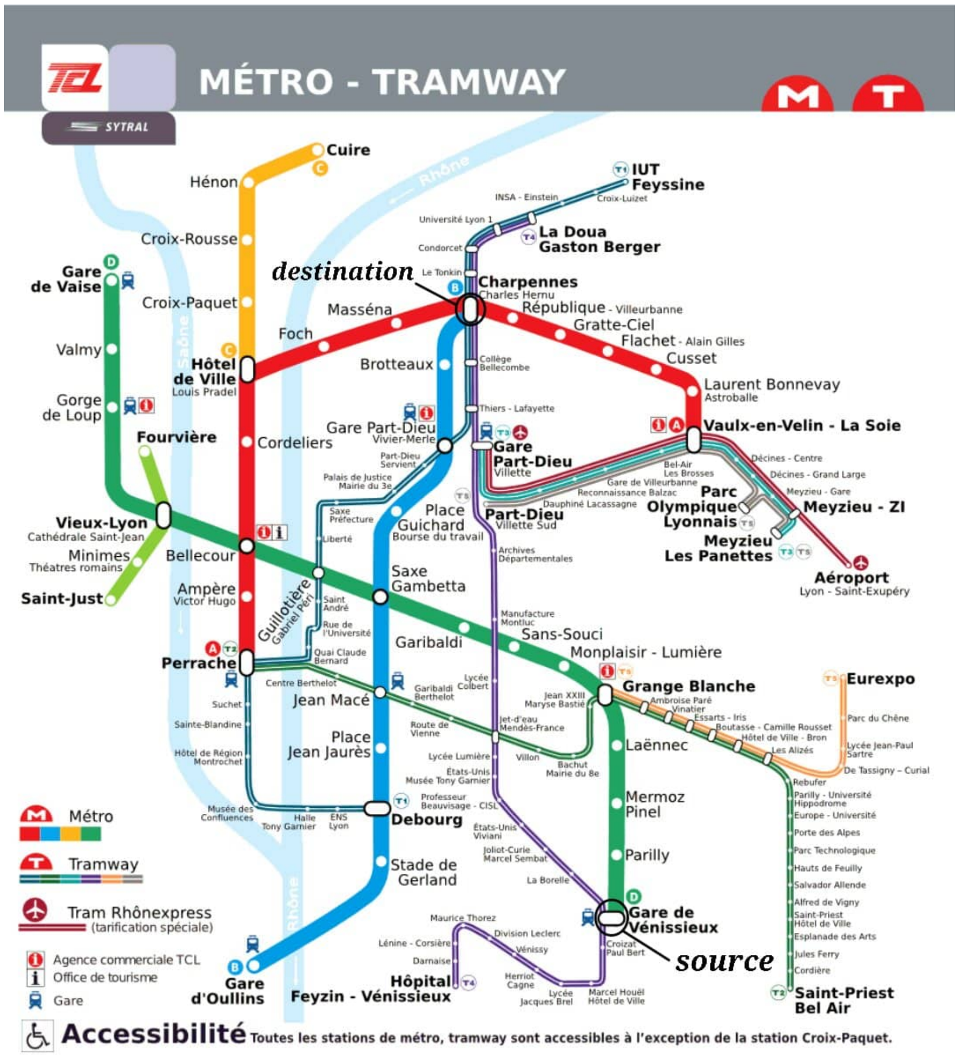


Figure 4.2: TCL lyon metro/tramway transport network

Table in 4.3 shows informations about test data and results, ACO parameters used are the same in 4.2, in addition to that, the number of itiration $i = 30$ and number of ants $m = 50$.

ACO parameters		Results (time, distance)		Excution time (ms)	
s	d	good	average	good	average
gare de vnissieux	charpennes	(87, 63.1)	(115, 86.1)	725.07	745
grange blanche	charpennes	(95, 74.6)	(115, 69.71)	534.14	558
debourg	charpennes	(57, 38.20)	(57, 38.20)	482.59	494

Table 4.3

4.4.2 Pseudorandom generated transport networks

Tabel 4.4 shows results of tests on pseudorandom generated transport networks, ACO parameters used are the same in 4.2, in addition to that, the number of itiration $i = 30$ and number of ants $m = 50$.

Graph info	Results (time, distance)		Excution time (ms)	
G(N,E)	good	average	good	average
G(20,83)	115, 86.1	295, 135.5	211.74	232
G(30,176)	664.74, 55.22	609, 78.2	309.08	337
G(50,315)	549.15, 72.73	702, 74.7	490.25	517
G(100, 740)	1155.60, 132.96	1457, 165.6	1015.53	1070

Table 4.4: ACO-MCSP applied on pseudo random generated multimodal transport networks

Conclusion

Algorithms based on the metaheuristic of ant colony do not guarantee finding an optimal solution in all possible cases. The construction of the presented ACO-MCSP algorithm reflects this particular approach through the proposal and the discussion of various variants of the execution of each of the elements of the procedure. It is possible to improve the method for the solution of the MCSP problem to approach or reach optimal solutions. An evaluation of the duration time and the quality of returned solutions will provide information for making a decision on the implementation of a given scheme as being of optimum quality or an alternative to more time-consuming procedures or procedures with higher computational cost. A future perspective to improve the ACO-MCSP is to focus on more criteria and work on procedure to regulate values of deposited pheromone to match graph values.

Bibliography

[1] Keith Clark Dorian Gaertner. On optimal parameters for ant colony optimization algorithms. *International Conference on Artificial Intelligence*, 2005.

[2] Marco Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, 1992.

[3] Heike TrautmannGnter RudolphKathrin KlamrothOliver SchtzeMargaret WiecekYaochu JinChristian Grimme. *Evolutionary Multi-Criterion Optimization, publisher = Springer, edition = 1.* 2004.

[4] Brian Slack Jean-Paul Rodrigue, Claude Comtois. *The Geography of Transport Systems, publisher = Routledge, edition = 3.* 2006.

[5] Thomas Stutzle Marco Dorigo. *Ant Colony Optimization, publisher = MIT Press, edition = 1.* 2004.

[6] Przemysaw Nowak Bartosz Musznicki Mariusz Glabowski, Piotr Zwierzykowski. Shortest path problem solving based on ant colony optimization metaheuristic. 2012.

[7] Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. 1987.

[8] Juliana Verga, Ricardo Coelho, and Akebo Yamakami. *Multimodal Transport Network Problem: Classical and Innovative Approaches.* 07 2018.

[9] Haicong Yu and Feng Lu. A multi-modal route planning approach with an improved genetic algorithm. *Adv. Geo-Spatial Inform. Sci*, 38, 01 2012.