POLITECNICO DI TORINO

MASTER's Degree in MANAGEMENT ENGINEERING

MASTER's Degree Thesis

Solution approaches for Vehicle Routing Problems of a set of small food shops in a local area of Torino

Supervisors

Prof. FABIO GUIDO MARIA SALASSA

Prof. FEDERICO DELLA CROCE DI DOJOLA

Candidate

ANDREA BARI

April 2021

Summary

This dissertation is the result of a collaboration with LINKS Foundation that aims to optimise the delivery service of a set of small food shops in a specific area of Torino. Each one of these businesses is currently managing a number of low volume deliveries on its own. I started my research by understanding the issues that these shops face everyday with the current system of delivery: owning a vehicle dedicated to deliveries results in costs that are too high, given the low quantity of orders for each store. As many micro and small enterprises in the food sector are facing the competition of bigger companies in terms of infrastructure costs, the reorganisation of the goods' distribution chain in a collaborative way (shared vehicles, spaces, human resources, processes, data and information) can lead to an improvement of the current situation.

After having understood the needs of these businesses, I developed an algorithm with Python, able to calculate the routes of vehicles starting from a shared depot, merging the routes of different shops in order to allow a more efficient delivery service. I simulated 1350 scenarios with different properties which allowed me to gain insights about the advantages of a shared system of delivery services, such as the reduction of transportation costs and lower number of vehicles used, helping small businesses with an alternative proposition to the current situation. The advantages of such a system also apply to the final customers, that would benefit from a more efficient infrastructure, lower delivery costs, the possibility to access a digital ordering platform.

Finally, I noticed how this kind of organisation would result in a much lower number of vehicles on the streets, contributing to a decreased level of pollution for the city of Torino.

Table of Contents

List of Figures

Chapter 1

Operations Research

1.1 Introduction

Operations Research has been defined by INFORMS (institute for Operations Research and the management science): Operations research aims to provide rational bases for decision making by attempting to understand and structure complex situations and to use this understanding to predict the behaviour of systems and improve their performance. Much of this work uses analytical and numerical techniques to develop and manipulate mathematical. and computer models for organisational systems composed of people, machines and procedures[1].

Among other branches of science, operations research plays a key role in economic problems. Its first applications date back to the late 18^{th} century with G. Monge, who made studies of a transportation problem, and then also with F. Taylor, who made a study of production methods almost a hundred years later. However, the term Operations Research was coined in the military context in the United Kingdom in the late 1930s just before the Second World War. In particular it started with the study concerning the efficient use of a military instrument for detecting the position of objects such as aircraft and ships: the radar. It was precisely in 1937 that the British Royal Air Force began experiments for an air defence control system, based on the use of a radar station located on the east coast, at Bawdsey Research Station. From the moment they realised the enormous difficulty in efficiently managing the information received from the radar, the supervisor of the Bawdsey Research Station decided to propose the development of a research program for the operational aspects of the system and not just the technical aspects, which were already satisfactory, therefore a group of expert scientists from various disciplines was created, called the "OR Team", hence the term "Operational Research"[2]. With time and the spread of the personal computer, there was an increasing spread of operational research in various industrial fields. Some examples where

Operational Research techniques are used are:

- Finance:
  In this field to determine the amount to be invested or the way in which an investment is to be made, e.g. using mathematical models to select investments and the amount of each (portfolio) or to determine the price of certain financial derivatives;

- Warehouse stock management:
  Finding production and stock levels for efficient product management, identifying when and how many products should be ordered to reorder the warehouse in order to minimise resource management costs;

- Determination of personnel shifts:
  Problem aimed at minimising personnel costs and respecting constraints. It arises for example in the management of personnel in a hospital, a company or a train;

- Minimum vehicle path:
  Identifying the minimum path of a vehicle according to the demands to be met, this in particular will be the main macro topic of the thesis, in which I will study a specific case.

It is distinguished by a strong interdisciplinary, mainly mathematics, computer science, engineering, economics and finance. It can be divided into 3 groups[3]:

1. Optimisation:
   (what-is-best approach) it formalises the problem in a mathematical model and identifies an optimal or sub-optimal solution for it;

2. Simulation:
   (what-if approach) it Formalises the problem in a mathematical model and determines "good" parameters using statistical or game theory methods;

3. Stochastic process:
   It creates probabilistic models in order to determine the behaviour of systems; it is the basis of financial engineering.

1.2 Modeling approach

Operations Research is therefore a methodology that cuts across many disciplines, applicable in many contexts where analytical methods are needed to improve the effectiveness of solutions. One of the branches is the modeling approach.

The term model is usually used to refer to an artificial construction made to highlight specific properties of real objects. The models are abstract, namely mathematical models that use the symbolism of algebra to highlight the main relations of the object to be modelled. The models and are made up of a set of relations that describe in a simplified but rigorous way one or more phenomena of the real world. The notion of a mathematical model to represent the real world is not new: This time was older, already in the 4^{th} century B.C. Pythagoras tried to build a mathematical model of the Universe.[2]

The interest for mathematical modelling has grown considerably and nowadays it is considered that through mathematical models it is possible to represent many aspects of the real world and to analyse its properties. This has led, as previously explained, to a huge development of the applications of mathematical modelling also outside the traditional applications to the mathematical-physical sciences, in fields like, for example, social and environmental sciences. As well as concrete examples, think of studies on the spread of epidemics as we are seeing with the current situation of Covid-19, or on environmental recovery, another highly studied topic. It is clear that in many cases the situations that we want to represent with a model are very complex and sometimes influenced by phenomena. For this reason, several classes of mathematical models have been defined:

- Stochastic models:
  They consider quantities that can be influenced by random phenomena, hence in a situation where uncertainty is present. «The word stochastic comes from the Greek word stokhazesthai meaning to aim or guess. In the real word, uncertainty is a part of everyday life, so a stochastic model could literally represent anything»[4].
  The stochastic models have a chance to give different outcomes every time the model is run in order to find a solution.

- Deterministic models:
  They consider exact quantities, hence when the prediction has 100% of accuracy, opposed to the randomness of the stochastic models.
  Deterministic models always have a set of equations that describe the system behaviour and always have exact outputs.

Furthermore, depending on whether the interactions between the quantities are immediate or distributed in time, we speak of static models and dynamic models. The modelling approach to solve a decision problem or, more generally, the use of mathematical methods for the solution of application problems, is usually carried out through various phases, as a resolution step, which are:

1. Analysis of the problem:
   It is necessary to determine the objectives to be achieved and the constraints

that the problem under consideration has. This initial phase is fundamental to the study and collection of the data in order to identify the logical-functional connections;

2. Construction of the model:
   Once the data have been classified, it is necessary to create the mathematical model, i.e. the representation of the real situation expressed through the use of mathematical expressions. There will always be an objective function to maximise (e.g. revenues or profits) or minimise (e.g. costs or staff); this function will depend on one or more variables to determine their respective values (e.g. sales quantity or number of staff to be employed). All this will be constrained in the model by equations and inequalities representing the properties of the problem;

3. Model analysis:
   In this case the intent is to deduct by analytical way, with reference to certain classes of problems, some important properties. The main ones are:

   - Existence of the optimal solution;
   - Conditions of optimality, that is an analytical characterisation of the optimal solution;
   - Stability of the solutions when the data or any parameters are changed.

4. Finding a solution:
   Once the mathematical model has been formulated, the possible optimal solution is sought, usually using using appropriate calculation algorithms. Since a model is never a perfect representation of the real problem, the solution may not be the best solution to the problem;

5. Checking the model and the solution:
   Once the optimal solution has been found in the model, it must be compared with reality and evaluated. The validation of the model can be done through experimental verification or simulation methods. Then it can be iterated all the process in order to refine and improve the solution[2].

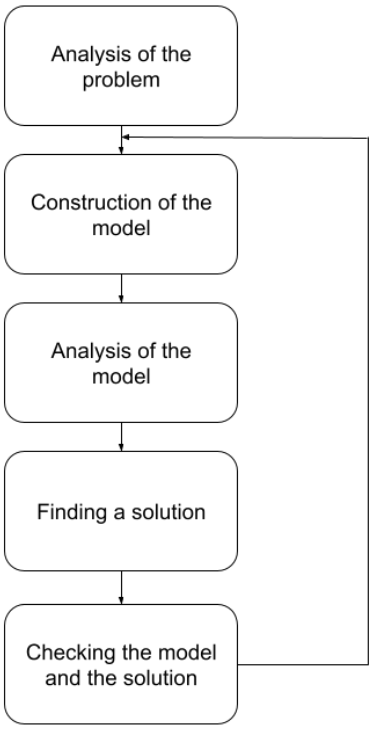


Figure 1.1: Modeling approach

1.3 Optimization models

«Many important and practical problems can be expressed as optimization problems. Such problems involve finding the best of an exponentially large set of solutions. It can be like finding a needle in a haystack. The obvious algorithm, considering each of the solutions, takes too much time because there are so many solutions. Some of these problems can be solved in polynomial time using network flow, linear programming, greedy algorithms, or dynamic programming. When not, recursive backtracking can sometimes find an optimal solution for some instances in some practical applications. Approximately optimal solutions can sometimes be found more easily. Random algorithms, which flip coins, sometimes have better luck. However, for the most optimization problems, the best known algorithm require $2^{O(n)}$ time on the worst case input instances. The commonly held belief is that there are no polynomial-time algorithms for them (though we may be wrong). NP-completeness helps to justify this belief by showing that some of these problems are universally hard amongst this class of problems. I now formally define this

class of problems.[5]»

Optimization modeling is a branch of mathematics which has the goal to determine the optimal maximum or minimum value of a set of complex equations. A key factor is that constraints such as time limitations (maximum of 8 hours per day), which are used to find a realistic solution, must be respected. Mathematical optimization makes use of techniques to evaluate complex models that most of the time are related to real-life business problems, such as scheduling, portfolio management, routing problems, and more.

In general terms, given a set $X \subseteq R^n$ and a function $f : X \to R$ and n is the number of real variables, an Optimisation problem can be formulated in the form:

$$\begin{cases} minf(x) \\ x \in X \end{cases} \tag{1.1}$$

Conventionally it is defined the minimum but it is completely indifferent in fact:

$$\begin{cases} maxf(x) \\ x \in X \end{cases} \tag{1.2}$$

Which is the same as:

$$\begin{cases} min - f(x) \\ x \in X \end{cases} \tag{1.3}$$

Therefore we can say that an Optimization problem consists in determining a minimum point of the function $f(x)$ among the points of the set X, if there exists at least one solution. Optimisation problems are often called, with equivalent terminology, Mathematical Programming problems. The admissible set X is the set of the possible solutions of the problem, which is a subset of R^n, hence $x = (x_1, x_2, ..., x_{\mathrm{n}})^T$ is an n-dimensional vector variable. A point $x \in X$ is called a feasible solution. The function f called objective function is composed of n real variables f(x_1, x_2, . . . , x_{n}). The optimization problem has different types of solutions[2]:

1. Feasible:
   The optimization problem is feasible if $X \neq 0$ therefore there is at least one admissible solution x inside the set X;

2. Unfeasible:
   The optimization problem is unfeasible if $X = 0$ therefore there are no possible solutions x inside the set X;

3. Unbounded:
   The optimization problem is unbounded above or below respectively for maximum and minimum problems if for any value $A > 0$ there exists a point $x \in X$ such that $f(x) > A$ or $f(x) < -A$. In other words when the solution tends to plus or minus ∞ and so it means that there is no bound for the set of feasible solutions$X$and thus the optimal solution $\rightarrow \pm\infty$.
   An example for both the cases is $f(x) = x^3$ in the first case we have $X = \{x : x \geq 3\}$ and we want to find the maximum, but we can see that for$X$tends to ∞ the function $f(x)$ tends to ∞, so it is unbounded above.
   In the case of min $f(x) = x^3$ and $X = \{x : x \leq 3\}$ we have the same x tends to $+\infty$ and the function $f(x)$ tends to -∞, so it is unbounded below;

4. Optimal:
   the optimization problem $(minf(x))$ has an optimal value $f(x^*)$ if there exists an $x^* \in X$ such that $f(x^*) \leq f(x)\ \forall x \in X$. The point x^* is called optimal solution or global minimum and the corresponding value $f(x^*)$, as said before, is called optimal value. In case we have $Maxf(x)$ if for a point x^* $f(x^*) \geq f(x)$ $\forall x \in X$.
   One example for $min : f(x) = 2 \times x$ and optimal solution $(x^*) = 0$.
   One example for $max : (x) = -2 \times x^2$ and optimal solution $(x^*) = 0$.

To conclude "Solving" an optimization problem therefore means, in practice:

- Determine if the feasible set is non-empty, or to deduce that there are no feasible solutions;
- Establish if there are optimal solutions, or to show that the problem does not have optimal solutions;
- Determine an optimal solution.

Types of problems

1. Continuous:
   In continuous optimization problems the variables used can assume all the real values $x \subseteq R^n$. Furthermore we have 2 types of continuous optimisation problems:
   - Constrained: when we have $X \subseteq R^n$.
   - Unconstrained: when we have $X = R^n$.

2. Discrete:
   Discrete Optimization problems consist in a set of problems with variables that can assume only values on a finite set. Also here we can divide this class in 2 other categories:

- Integer programming: when the variables of set X can take only integer values so $X \subseteq Z^n$.
- Combinatorial programming: when the variables can take boolean values so $X \subseteq \{0,1\}^n$.

3. Mixed:
   Mixed problems have some continuous variables and some of them can be integer or boolean.

Creating the model of the optimization problem is a step really important to understand all the characteristics of the problem we want to solve. There is the feasible set X, which is described by a finite number of inequalities of the type $h(x) \leq 0$, where h is a function defined on R^n with real values $a_i \in R$. Formally, given m functions $h_i : R^n \to \mathrm{R}$, $i = 1, ..., m$ we express X in the form:

$$X = \{x \in R^n \quad | \quad h_1(x) \leq a1, ... \quad , h_m(x) \leq am\} \tag{1.4}$$

Every inequality $g_i(x) \leq 0$ is called constraint and the admissible set is then formed by all those points $x \in R^n$ that are solutions of the system of inequalities.:

$$\begin{cases} h_1(x) \leq a_1 \\ h_2(x) \leq a_2 \\ ... \\ h_m(x) \leq a_m \end{cases} \tag{1.5}$$

In this formulation of the set X the constraints are in the form of less than or equal, but it is possible to have different cases in which the constraints are expressed with inequality constraints in the form of greater than or equal and equality constraints. For instance a greater than or equal constraint of the type $h(x) \geq 0$ can always be transformed into a less than or equal constraint by changing the form into $-h(x) \leq 0$.

Another example is an equality constraint $h(x) = 0$ which can be rewritten in the equivalent form of the two inequalities $h(x) \leq 0$ and $-h(x) \leq 0$.

In conclusion the optimization problem can be formulated in the Mathematical Programming form:

$$\begin{cases} minf(x) \\ h_i(x) \leq 0 \quad i = 1, ..., m \end{cases} \tag{1.6}$$

The problems of Mathematical Programming can be classified according to the properties of the objective function and of the constraints taking into consideration, among the most significant, linearity and convexity. A first distinction is the one that refers to the hypothesis of linearity. From this point of view, we can distinguish[2]:

- Problems of Linear Programming (LP): In this case the objective is a linear function of the type: $c_1x_1 + c_2x_2 + ... + c_nx_n$.
  The constraints are expressed by a system of linear inequalities in the form: $a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \geq b_i$;
- Non-linear programming (NLP) problems:
  In this case the objective or some of the constraints are not linear, for example: $x_i \times y_j$.

The model is constructed by an objective function and the constraints with the appropriate variables:

- Objective function:
  It is used to measure a particular aspect of a system with the intent to minimize or maximize it, finding the optimal values of the variables;
- Variables:
  They represent the components of the system, which are used to determine the objective function and the constraints;
- Constraints:
  They are used to limit the values of the objective function corresponding to some characteristics as for example, working at least 8 hours per day, where the hours is a variable of the system.

Now I will show some examples of linear and non-linear programming:

1. Linear programming:

$$\begin{cases} min \quad x_1 + 3x_2 \\ x_1 + 2 \times x_2 \geq 10 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases} \tag{1.7}$$

   It is linear since all the functions in the objective function or in the constraints are linear;

2. Non-linear programming:

$$\begin{cases} min \quad x_1 + 3x_2 + x_1 \times x_2 \\ x_1 + 2x_2 \geq 10 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases} \tag{1.8}$$

   In this case it is non-linear programming because in the objective function x_1 and x_2 are multiplied between them.

A simple example of a construction Linear Programming Problem is: A company S processes oil into aviation fuel and heating oil. It costs 40 to purchase each 1,000 barrels of oil, which is then distilled and yields 500 barrels of aviation fuel and 500 barrels of heating oil. Output from the distillation may be sold directly or processed in the catalytic cracker. If sold after distillation without further processing, aviation fuel sells for 60 per 1,000 barrels, and heating oil sells for 40 per 1,000 barrels. It takes 1 hour to process 1,000 barrels of aviation fuel in the catalytic cracker, and these 1,000 barrels can be sold for 130. It takes 45 minutes to process 1,000 barrels of heating oil in the cracker, and these 1,000 barrels can be sold for 90. Each day, at most 20,000 barrels of oil can be purchased, and 8 hours of cracker time are available. If we want to find the optimal daily profit for company S we can write the mathematical model to find the solution.

Variables:

- x_1 is the number of 1000 barrels of oil bought;
- x_2 is the number of barrels of aviation fuel processed in the cracker;
- x_3 is the number of barrels of heating oil processed.

Constraints:

- 8 hours of cracker time are available: $x_2 + 0.75x_3 \leq 8$;
- At most 20,000 barrels of oil can be purchased: $x_1 \leq 20$;
- Distilled and yields 500 barrels of aviation fuel: $0.5x_1 - x_2 \geq 0$;
- Distilled and yields 500 barrels of heating oil: $0.5x_1 - x_3 \geq 0$;
- All the barrel units are positive: $x_1 \geq 0$, $x_2 \geq 0$, and $x_3 \geq 0$.

So the final formulation is:

$$\begin{cases} max \quad 60(0.5x_1 - x_2) + 130x_2 + 40(0.5x_1 - x_3) + 90x_3 - 40x_1 \leq 20 \\ x_2 + 0.75x_3 \leq 8 \\ 0.5x_1 - x_2 \geq 0 \\ 0.5x_1 - x_3 \geq 0 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases} \tag{1.9}$$

The optimal solution to this problem is:

- $x_1 = 20$

- $x_2 = 8$
- $x_3 = 0$

The corresponding total profit is: 760.

1.4 Linear Programming

«The maximum flow and minimum cut problems are examples of a general class of problems called linear programming PL. Many other optimization problems fall into this class, including minimum spanning trees and shortest paths, as well as several common problems in scheduling, logistics, and economics. Linear programming was used implicitly by Fourier and Jacobi in the early 1800s, but it was first formalized and applied to problems in economics in the 1930s by Leonid Kantorovich. Kantorivich's work was hidden behind the Iron Curtain (where it was largely ignored) and therefore unknown in the West. Linear programming was rediscovered and applied to shipping problems in the late 1930s by Tjalling Koopmans. The first complete algorithm to solve linear programming problems, called the simplex method, was published by George Dantzig in 1947. Koopmans first proposed the name "linear programming" in a discussion with Dantzig in 1948. Kantorovich and Koopmans shared the 1975 Nobel Prize in Economics 'for their contributions to the theory of optimum allocation of resources'. Dantzig did not; his work was apparently too pure. Koopmans wrote to Kantorovich suggesting that they refuse the prize in protest of Dantzig's exclusion, but Kantorovich saw the prize as a vindication of his use of mathematics in economics, which his Soviet colleagues had written off as "a means for apologists of capitalism"»[6].

A linear program is composed by:

- A single linear objective function to minimize or maximize:

$$f(x_1, ..., x_m) = c_1x_1 + ... + c_mx_m = \sum_{j=1}^{m} c_jx_j \tag{1.10}$$

- A set of linear inequalities:

$$\begin{cases} a_{11}x_1 + ... + a_{1m}x_m \leq b_1 \\ ... \quad + ... + \quad ... \quad \leq ... \\ a_{(n-1)1}x_1 + ... + a_{(n-1)m}x_m \leq b_{n-1} \\ a_{n1}x_1 + ... + a_{nm}x_m \leq b_n \end{cases} \tag{1.11}$$

Which is equivalent to:

$$\sum_{j=1}^{m} a_{ij}x_j \leq b_i \quad \forall i = 1...n \tag{1.12}$$

In linear algebra it is possible to use a different notation:

$$\begin{cases} min \quad c^T x \\ Ax \leq b \end{cases} \tag{1.13}$$

In this case we have the m-vector $c = (c_1, ..., c_m)^T$, the m-vector of variables $x = (x_1, ..., x_m)^T$ then $A = (a_{ij})$ is a matrix (nxm) and the vector $b = (b_1, ..., b_n)^T$. A linear programming problem to be in canonical or standard form it has to be in one of the following forms[7]:

- Standard form:

$$\begin{cases} max \quad c^T x \\ Ax = b \\ x \geq 0 \end{cases} \tag{1.14}$$

- Canonical form:

$$\begin{cases} max \quad c^T x \\ Ax \leq b \\ x \geq 0 \end{cases} \tag{1.15}$$

Any linear program can be converted into standard or canonical form, depending to what form to use. I will explain the cases to standard form, which is similar to conversion into canonical form:

- If it is $minf(x)$ replace it with $Max - f(x)$
- In the case there is a free variable x_i which can be negative (profit/loss). it is possible to change the variable with the substitution method:
  $x_i = x_i^+ - x_i^-$
  $x_i^+ \geq 0$
  $x_i^- \geq 0$
- For inequality constraints, for example with $x_1 + x_3 \leq 10$ we can add one non-negative variable $z \geq 0$ and the inequality becomes $x_1 + x_3 + z = 10$

1.5 Integer Linear Programming

The problems of Integer Linear Programming (PLI) differ from those of PL only for the fact that the variables can take only integer values. This integrity constraint has, however, an enormous impact. Integer variables can be used to model logical conditions and situations in which decisions are made among a finite number of possible alternatives. It can be said that the vast majority of models used in

practice are PLI, since in most real-world applications logical conditions exist and discrete choices must be made. Many combinatorial optimization problems CO, which we will explain later, can be formulated as PLI problems. In fact, in practice we tend to consider substantially coincident the two classes of problems[7]. This is due to two concomitant reasons:

- CO problems are normally formulated as PLI problems, and much of the solution approaches for CO problems are based on such formulations;
- almost all efficient techniques for solving PLI problems are based on the identification and exploitation of specific combinatorial structures in the PLI model, i.e., on the identification of sub CO problems corresponding to the PLI model or parts thereof.

There is therefore a strong connection between the problems of CO and those of PLI. The two classes are not however completely coincident. On the one hand, PLI provides a powerful language to formulate in a uniform way both CO problems defined on very different structures, but also problems that can be very difficult to relate to CO problems: PLI is in some sense "more expressive" than CO. On the other hand, there may exist many different PLI formulations of the same CO problem, and the formulation as a CO problem is often "more informative" than the PLI formulations, in the sense that it may be easier to derive useful properties for the solution of the problem by working directly on its combinatorial structure rather than on its formulations in terms of PLI.

Since we can apply to PLI problems the same transformations we saw in before for PL problems, we can assume that PLI problems are expressible in standard forms analogous to those already introduced, for example:

$$(PLI) \quad max\{cx : Ax \leq b, x \in Z^n\} \tag{1.16}$$

We also speak of Mixed Linear Programming (PLM) problems when only some of the variables are bound to be integer, such problems then have the form:

$$(PLM) \quad max\{c'x' + c'x'' : A'x' + A''x' \leq b, x' \in Z^n\} \tag{1.17}$$

Almost all of the approaches for PLI that we will describe can be generalized to PLM, often only at the cost of complications in description.

Chapter 2

Combinatorial optimization

2.1 Algorithm complexity

Once a problem P has been formulated, it must be solved; therefore, people are interested in the development of computational tools, i.e., algorithms, that given any instance p of the problem are able to provide a solution in a finite time. An algorithm that solves P can be defined as a finite sequence of instructions that, applied to any instance p of P, will stop after a limited number of steps (i. e., elementary computations), providing a solution of p or indicating that p has no feasible solution.

Generally one is interested in finding the most efficient algorithm for a given problem. In order to study algorithms from the point of view of their efficiency, or computational complexity, it is necessary to define a computational model; some examples of computational models are the Turing Machine and the Register Machine (RM).

Given a problem P, an instance p, and an algorithm A that solves P, we denote by cost (or complexity) of$A$applied to p a measure of the resources used by the computations that A performs on a machine M to determine the solution of p. Resources, in principle, are of two types: occupied memory and computation time. Very often the most critical resource is the computation time (time of execution of the algorithm) and therefore it is used primarily as a measure of the complexity of the algorithms. Assuming that all elementary operations have the same duration, the computation time can be expressed as the number of elementary operations performed by the algorithm. Given an algorithm, It is desirable to have a measure of complexity that allows an evaluation of its effectiveness and possibly a comparison with alternative algorithms. Knowing the complexity of A for each of the instances of problem P is not possible (the set of instances of a problem is normally infinite), nor would it be of practical use.

So we try to express the complexity as a function $g(n)$ of the dimension n of the instance to which the algorithm is applied, defined as a measure of the number of bits needed to represent, with a reasonably compact encoding, the data defining the instance, i.e. a measure of the length of its input. Since for each dimension there are in general many instances of that dimension, we choose $g(n)$ as the cost required to solve the worst case among the instances of dimension n. At this point the function $g(n)$ turns out to be sufficiently rigorously defined, but it continues to be difficult to use as a measure of complexity, if not practically impossible, to evaluate $g(n)$ for any given value of n. This problem is solved by replacing the function $g(n)$ with its order of magnitude; we then speak of asymptotic complexity. Given two functions $f(x)$ and $g(x)$, we will say that:

1. $g(x)$ is $O(f(x))$ if there exist two constants c_1 and c_2 for which $g(x) \leq c_1 f(x) + c_2 \ \forall x$;
2. $g(x)$ is $\Omega(f(x))$ if $f(x)$ is $O(g(x))$;
3. $g(x)$ is $\Theta(f(x))$ if $g(x)$ is both $O(f(x))$ and $\Omega(f(x))$.

Let $g(x)$ be the number of elementary operations that are performed by the algorithm A applied to the most difficult instance, among all those having input length x, of a given problem P. We will then say that the complexity of A is an $O(f(x))$ if $g(x)$ is an $O(f(x))$, is an $\Omega(f(x))$ if $g(x)$ is an $\Omega(g(x))$ and a $\Theta(f(x))$ if $g(x)$ is a $\Theta(f(x))$. An algorithm is said to have polynomial complexity if the execution time is of order $O(x^k)$, where k is a constant independent of the input length x. If the time complexity function cannot be bounded by a polynomial, the algorithm is said to have exponential complexes. If the expression b^l, where l is a constant and b is the largest input value, is part of the limiting function, i.e. if the running time is of order $O(x^k b^l)$, then the algorithm is said to have pseudo-polynomial complexity[8].

2.1.1 Classes P and NP

To make a more rigorous classification of the different problems, it is convenient to refer to problems in decision form. A first important class of problems is the class NP, consisting of all decision problems whose problem can be solved in polynomial time. In other words, the problems in NP are those for which it is possible to verify efficiently an answer yes, because it is possible to decide in polynomial time if a solution x is feasible for the problem. In other words, problems in NP are those for which there exists a polynomial-length computation that can lead to constructing a feasible solution, if one exists, but this computation may be hidden within an exponential set of similar computations among which one does not know how to discriminate.

A subset of the class NP is the class P, consisting of all problems solvable in polynomial time, i.e. containing all those decision problems for which there are algorithms of polynomial complexity that solve them. For this reason, problems in P are also called polynomial-time problems. Clearly $P \subseteq NP$, but a particularly important question is whether there are problems in NP that do not also belong to P, that is, whether $P \neq NP$. This question cannot be answered, although it is highly probable that the answer is positive, i.e. that it is indeed $P \neq NP$[9].

2.1.2 NP-Complete and NP-Hard Problems

Given two decision problems P and Q, and assuming the existence of an algorithm A_Q that solves Q in constant time, we will say that P reduces in polynomial time to Q, and write $P \propto Q$, if there exists an algorithm that solves P in polynomial time using A_Q as a subprogram. In such a case we speak of polynomial time reduction of P to Q.

The relation $\propto$ has the following properties :

1. It is reflexive: $A \propto A$;
2. It is transitive: $A \propto B$ and $B \propto C \rightarrow A \propto C$;
3. $A \propto B$ and $A \notin P \rightarrow B \notin P$;
4. $A \propto B$ and $B \in P \rightarrow A \in P$.

We can now define the class of NP-complete problems: a problem A is called NP-complete if $A \in NP$ and if for every $B \in NP$ we have that $B \propto A$. The class of NP-complete problems constitutes a subset of the NP-class of particular importance. NP-complete problems have the important property that if there exists for one of them a polynomial algorithm, then necessarily all problems in NP are solvable in polynomial time, and hence $P = NP$. In a sense, such problems are the hardest of the problems in NP. A problem is said to be NP-complete in a strong sense if no pseudo-polynomial algorithm exists for it unless $P = NP$.

A problem that has an NP-complete problem as its special case is called an NP-hard problem and has the property of being at least as difficult as NP-complete problems (unless there is a polynomial multiplicative function). Note that an NP-Hard problem may also not belong to the NP-class. As defined for NP-complete problems, similarly a problem is said to be NP-hard in a strong sense if no pseudo-polynomial algorithm exists for it unless $P = NP$.

2.2 Introduction to Combinatorial optimization

Combinatorial optimization deals with discrete type objects such as graphs or sets and it studies problems that consist in arranging, grouping, ordering these objects in an optimal way. Therefore such problems always have a finite number of solutions, which would imply a certain ease in finding the optimal solution (at least by comparing the solutions one by one) but this is by no means true: in most cases the number of solutions is, in fact, very high and therefore the need arises to design algorithms suitable for the various types of problems, then evaluating their efficiency [7].

In the case in which an optimization problem is characterized by the fact that in each instance the realizable region F contains a finite number of points (and therefore the optimal solution can be found by comparing a finite number of solutions), we speak of Combinatorial Optimization (or Discrete Optimization) problem; as we said before we speak instead of Continuous Optimization problem if the realizable region can contain an uncountable infinity of points. While in the Combinatorial Optimization models the variables are constrained to be integers, in the Continuous Optimization models the variables can assume all real values. There are two types of solution methods for Combinatorial optimisation problems (COP):

- Exact methods;
- Heuristic methods.

2.2.1 Exact methods

Exact methods can determine an optimal solution: a feasible solution that optimises (minimises or maximises) the objective function. The heuristic methods provide a feasible solution that is not certain to be optimal, in fact it is difficult to find the optimum with this type of method, but a good approximation can be achieved [10].

In some cases, it is possible to find "efficient" exact algorithms to solve this kind of problems: for example, the problem of finding the shortest paths on a graph, under some reasonable assumptions often encountered in practice, can be solved by Dijkstra or Bellman-Ford algorithms, which are able to find optimal solutions in reasonable time.

Some examples of exact methods will be presented in the following:

- The branch-and-bound method;
- The cutting plane algorithms;
- The dynamic programming methods.

Branch-and-bound

The branch-and-bound method is the most commonly implemented method in commercial software for PLI and is based on the idea of an implicit enumeration of feasible solutions to a problem. This method consists of an algorithm that, starting from the linear relaxation of an integer linear program, comes to find an integer optimal solution of the original problem by splitting it into several distinct sub problems, that is, by splitting into several subsets the set of all feasible solutions for the problem, by means of the choice of the values of one of the decision variables. Given a PLI problem, which we assume to be minimization and in standard form, $min\{c^T x : Ax = b, x \geq 0, x \in Z\}$ and the polyhedron associated with its linear relaxation $P = x : Ax = b, x \geq 0$.

The scheme on which branch-and-bound methods are based is as follows:

1. The linear relaxation of the original problem is solved by determining $x^* \in argmin\{c^T x : x \in P\}$;

2. If x^* is integer, then it is also the optimal solution of the problem;

3. Otherwise, let ${x_j}^*$ be a non integer component of x^*. The original problem is equivalent to the following problem:

   $min\{c^T x : x \in P \cap Z \cap (\{x : x_j \leq \lfloor {x_j}^* \rfloor\} \cup \{x : x_j \geq \lceil {x_j}^* \rceil\})\}$;

4. Two sub problems are originated from the original problem:

   - $min\{c^T x : x \in P, x \leq \lfloor {x_j}^* \rfloor, x \in Z\}$;
   - $min\{c^T x : x \in P, x \geq \lceil {x_j}^* \rceil, x \in Z\}$.

   To them it comes applied, In recursive way, the same algorithm of the two possible found optimal solutions, that one of inferior value it is the optimal solution of the problem.

The execution of the method follows the path of a tree in which the nodes represent the variable and the branches correspond to the assignments of everyone of the possible values to the variable. Once found in the exploration of a zone of the tree of search a realizable solution for the problem, to verify if it is an optimal solution it is necessary to visit the remaining part of the space of search and to verify that better solutions do not exist, that is assignments of values to the variables that improve the value of the objective function regarding that of the found solution[8].

Cutting planes

Given a PLI problem, one method of obtaining its linear relaxation as close as possible to the ideal formulation of the problem is to add inequalities, called Cutting planes, to the set of constraints of the relaxation. According to this method, inequalities are added to the set of constraints of a linear relaxation that has a fractional optimal solution that hold on one hand the integer solutions of the problem and simultaneously are not satisfied by the fractional optimal solution of the current relaxation. It is then to add to the pre-existing system of constraints additional constraints that cut out the fractional solution (thus eliminating the part of the feasible region containing this solution) which as such can certainly not be an optimal solution of the original problem. This procedure, that is the addition of cutting planes to the relaxation of the original problem, iterates until an integer solution is reached, which will be the optimal solution of the problem.

Methods of this type were the first ones used for solving PLI problems, and are still the most important ones along with enumerating methods, such as the branch-and-bound method. Thus, the generation of cutting planes determines the insertion in the system of constraints of the relaxed problem of some linear inequalities that gradually remove the fractional solutions, unfeasible for the original PLI problem, without eliminating any integer solution[8].

Dynamic programming

Dynamic Programming (DP) is a mathematical technique that is often useful for making interrelated decisions and that allows one to tackle seemingly intractable problems, i.e., those with an apparent exponential complexity. It was originally introduced by Bellman in 1957 to solve some multi-stage decision problems and optimal control problems. Originally, DP was created for problems where a decision must be made for each instant of time, providing a systematic procedure for determining the combination of decisions that maximizes total efficiency.

DP solves computational problems by putting together the solutions of a number of sub problems, relying on the principle of optimality, according to which in some cases a properly decomposed problem can be solved to the optimum through the optimal solution of each sub problem. In this context the term programming does not mean a program written in some programming language, but a tabular solution method.

In general, DP is adopted to solve optimization problems but, unlike PL, there is no standard mathematical formulation of the DP problem; rather, it IS a general technique for solving certain problems, and the particular equations used must be developed and adapted in each individual situation. Algorithms based on DP divides the problem to be solved into a number of sub problems, even if not necessarily independent, and recursively solve these sub problems. In the case

of non-independent sub problems, the resolution requires solving common sub problems. In this case, the DP solves a problem only once and stores the solution in a table for use whenever the same problem occurs again[8].

2.2.2 Heuristic methods

For more complex problems, where "efficient" algorithms are not available, a possible approach may be to formulate the problem as a Mixed integer Linear Programming model and solve it with a MILP solver (Cplex, Gurobi, AMPL etc.), which makes use of general purpose exact algorithms that guarantee to find the optimal solution. It is not always possible to apply exact solution methods, due basically to two issues: complexity, and the time available to find the solution, which can be limited.

This is why the use of a heuristic method instead of an exact one is important: the complexity of a problem does not justify the use of heuristics, as the literature can provide valid exact algorithms. The use of heuristics is therefore motivated by the complexity together with the consideration of the appropriateness of implementing exact methods, the computational time available, the size of the instances to be solved etc. For example, it is always advisable to make an attempt to formulate the mathematical model: this effort is useful in the analysis phase of the problem, but also as an operational tool, since the increasing efficiency of solvers may make model implementation a viable approach to obtain an exact solution in reasonable run times.

The properties of the problem and/or the context of the solution may lead to the inappropriate application of exact methods , while it is necessary to provide "good" solutions that are feasible in "reasonable" time. There are some cases where a proved optimal solution is required, while in many other cases, including perhaps most real-world cases, a good approximate solution is sufficient, particularly for large instances of a problem. For example:

- For many parameters (data) that determine a Combinatorial optimisation problem, collected from a real application, only estimates are available, which may also be error-prone, and it may not be worth waiting long for a solution whose value (or even feasibility) cannot be guaranteed;
- It can be solved to provide a possible solution to a real problem, towards a rapid scenario evaluation (e.g. operational contexts, use of optimisation algorithms for support to business decisions);
- It can be declared in a real-time system, so that a "good" feasible solution is required to be provided within a limited time.

These examples attest to the extensive use of methods that provide good

enough solutions and guarantee acceptable computation times, even if they cannot guarantee optimality: they are called heuristic methods[10].

The main types of heuristics methods are:

- Constructive heuristics: are applicable if the solution can be obtained as a subset of some elements. In this case we start from an empty set and iteratively add one element at a time. For example, if the element is chosen on the basis of local optimality criteria, the so-called heuristics are realized greedy. An essential feature is the progressiveness in the construction of the solution;

- Metaheuristic methods: these are general methodologies, algorithmic schemes designed independently of the specific problem. These methods define components and their interactions in order to achieve a good solution. The components must be specialized for individual problems. Among the best known and most established metaheuristics are Local Search, Simulated Annealing, Tabu Search, Variable Neighborhood Search, Greedy Randomized Adaptive Search Techniques, Genetic Algorithms, Scatter Search, Ant Colony Optmization, Swarm Optmization, Neural Networks, etc.

- Approximate algorithms: these are heuristic methods with guaranteed performance it is possible to demonstrate formally that, for each instance of the problem, the solution obtained will not be worse than the optimum (possibly unknown) beyond a certain percentage;

- Hyper-heuristics: these are topics at the border with artificial intelligence and machine learning, in which the research is in a pioneering phase. In this case, the aim is to define algorithms that are able to discover optimization methods and adapt them automatically to different problems.

Chapter 3

Graphs

3.1 Introduction

Graphs are discrete mathematical structures used both for mathematics and for a wide range of application fields related to many real-world problems such as transport (GPS), computer science (computer networks, site maps) and even business organisation. In fact, they lend themselves to representing problems that could be apparently very different from each other in a simple and unified language. This serves to explain their importance in applied mathematics and, in particular, in Operations Research.

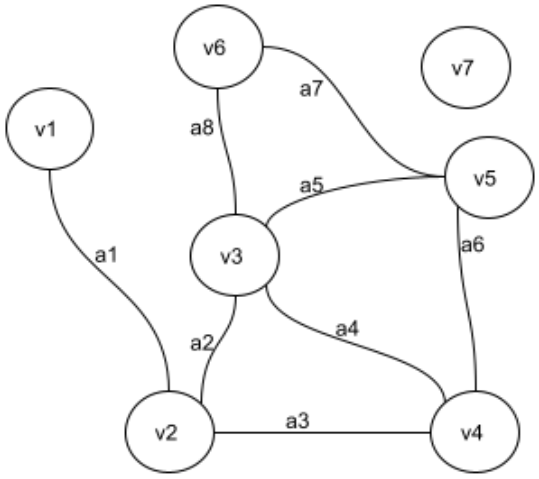


Figure 3.1: Generic graph

A graph, G=(V,A) is a pair of sets:

- V is a finite, discrete, non-empty set of elements.
- A is a finite set of pairs of discrete elements of V.

V is called a set of nodes, and is denoted by $V = \{v_1, v_2, ..., v_n\}$. The set A, called the set of edges, is in general indicated with $A = \{a_1, a_2, ..., a_m\}$ and is a subset of the Cartesian product $V \times V$, that is a set of vertex pairs of the graph $G = (V, A)$. Conventionally, the letters n and m indicate the cardinality of V and A respectively, in other words, the number of vertices and number of edges of the graph $G = (V, A) : n = |V|$ and $m = |A|$. An edge will be indifferently denoted either by a name that identifies it, for example a_k, or by a pair of nodes (v_i, v_j). Given the edge $a_k = (v_i, v_j)$ the nodes v_i and v_j are defined as extremes of a_k, and the side a_k, identified with the pair formed by its extremes (v_i, v_j), is said to be 'incident' upon v_i and v_j. A graph can be weighted when there is a number (weight) assigned to its edges, it is used to define distances, time travel, capacities, etc[11].

Graphs, in general, can be divided into two groups:

- Undirected graphs;
- Oriented graphs.

3.2 Undirected graphs

An undirected graph $G = (V, A)$ is defined by a finite set $V(G) = \{v_1, v_2, ..., v_n\}$ of elements called vertices and by a set $A(G) = \{a_1, a_2, ..., a_m\}$ of unordered pairs of nodes called edges[12]. Given any edge $a_k = \{v_i, v_j\}$, the connection between the two extremes is univocal, which means, it has the same value as the connection between v_i and v_j ($a_k = \{v_i, v_j\} = \{v_i, v_j\}$) and in this case we indicate the edges with the notation $\{v_i, v_j\}$ to highlight the fact that the two extreme nodes of the side are defined as an unordered pair[2].

Two vertices v_i, v_j are said to be adjacent if the side $\{v_i, v_j\}$ belongs to A and therefore connects them, while two edges are said to be adjacent if they have a vertex in common. The neighbourhood of a node v_i in G, denoted by $N(v_i)$, is the set of nodes adjacent to v_i.

In the case where $N(v_i) = 0$ v_i is said to be isolated. We define the star of v_i in G, indicated with $\delta(v_i)$, as the set of arcs 'incident' on v_i.

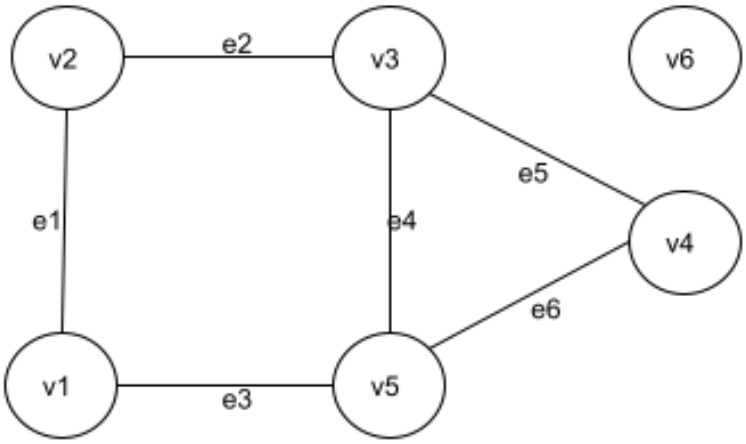


Figure 3.2: Undirected graph

Example of the graph $G = (V, E)$ represented in Figure 3.2, which is composed by the common 2 sets of elements:

- $V = \{v_1, v_2, ..., v_6\}$;
- $E = \{e_1, e_2, ..., e_6\}$ or $E = \{(v_1, v_2), (v_2, v_3), (v_1, v_5), (v_3, v_5), (v_3, v_4), (v_4, v_5)\}$.

Some examples[13] of the definitions written above related to Figure 3.2:

- $e_3 = (v_1, v_5)$,v_1 and v_5 are the extremes of e_3;
- v_3 and v_2 are two nodes adjacent having e_2 as arch in common;
- e_3 and e_1 are two arcs adjacent having v_1 as node in common;
- $N(v_5) = \{v_1, v_3, v_4\}$ is the neighbourhood of the node v_5, all the nodes adjacent to v_5;
- v_6 is an isolated node;
- $\delta(v_5) = \{e_3, e_4, e_6\}$ is a star of v_5 composed by the set of arcs 'incident' on v_5.

A subgraph of $G = (V, A)$ is defined as a graph $H = (W, E)$ such that $W \subseteq V$ and $E \subseteq A$. We call a subgraph induced by $W \subseteq V$ in $G = (V, A)$, the graph $H = (W, E)$, where the set of arcs E is such that the arc (v_i, v_j) belongs to E if and only if:

1. v_i and v_j belong to W;
2. $\{v_i, v_j\} \in A$.

We can say that the subgraph H inherits all the arcs of G whose extremities are both contained in the subset W.

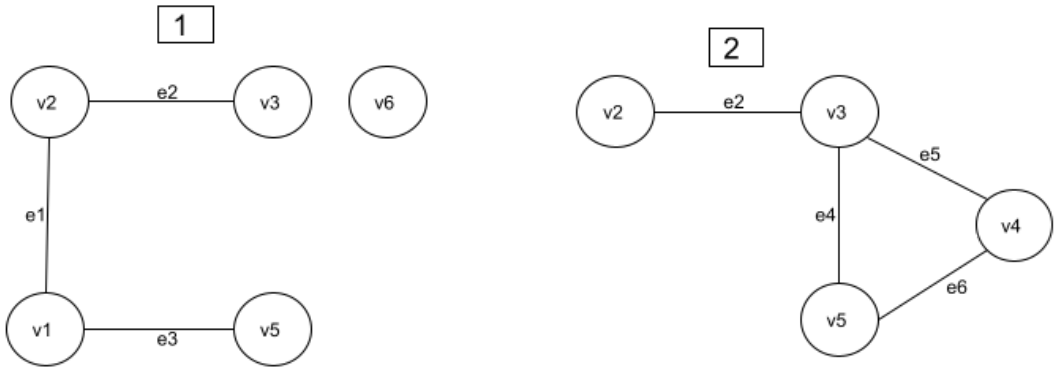


Figure 3.3: Subgraphs

Example in Figure 3.3 two subgraphs of the graph in Figure 3.2 are shown. There are two types of subgraph:

- The first one is not an induced subgraph ("missing" the arc e_4);
- The second one is induced in G by the set of nodes $\{v_2, v_3, v_4, v_5\}$.

The degree of vertex v_i, usually denoted by $d_g(v_i)$, is given by the number of sides 'incident' on vertex v_i itself. In the case where $d_g(v_i) = 0$ we find that v_i is an isolated node (see node v_6 in Figure 3.2) while in the case of $d_g(v_i) = 1$ we can define v_i as a hanging vertex (see node v_5 in Figure 3.2).

For each graph $G = (V, A)$ we can define a path, a sequence of distinct vertices $v_0, v_1, ..., v_n$ together with a sequence of sides connecting them $\{v_0, v_1\}$, $\{v_1, v_2\}$, ..., $\{v_{n-1}, v_n\}$. The vertices v_0 and v_n are called ends of the path. A walk is a path when it is allowed to use repetitions of nodes and/or edges. If the walk is closed it means its end and start coincide, and it is called a cycle or circuit. A graph is connected if and only if there is a path between any two random nodes of the graph.

"A tree is a graph that has no cycle and is connected. We define $R = (V', A')$ a spanning tree of a graph $G = (V, A)$ if $V' = V$, $A' \subseteq A$, and R is a tree. The Minimum spanning tree problem aims at finding a spanning tree of minimum weight. Given a graph $G = (V, A)$, a tour of G is a cycle passing through each vertex of the graph. A graph is Eulerian if it is connected and each node has even degree. An Eulerian tour of G is, for the edges, the equivalent of what a tour of G is for nodes: we pass through any edge exactly once"[14].

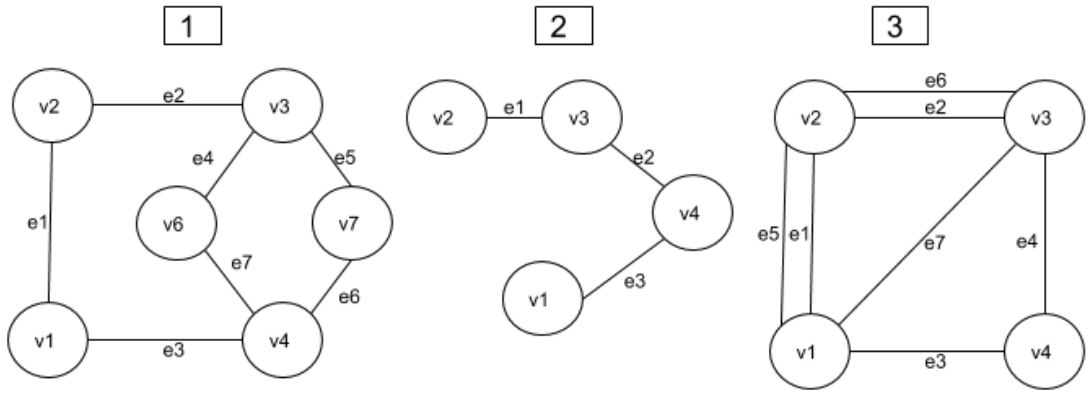


Figure 3.4: Path, tour, Eulerian tour

Other examples of new concepts introduced above linked to Figure 3.4:

- On the left there is a connected but not a tree due to the cycles, one of them is:(v_3, v_6, v_5, v_7);
- Always on the left an example of path between v_2 and v_7 is: (v_2, v_3, v_7);
- In the middle there is an example of a tree because there are not cycles in the graph;
- In the last one (right) there is an example of Eulerian graph since it is connected and the degree of each node is even;
- Another important concept, in the graph on the right, is a tour: (v_1, v_2, v_3, v_4);
- Finally always on the right an Eulerian tour of it is: $(v_1, v_2, v_1, v_3, v_2, v_3, v_4)$.

3.3 Oriented graphs

An oriented graph $G = (V, E)$ is composed by:

- A finite set $V(G) = \{v_1, v_2, ..., v_n\}$ of elements called vertices;
- A set $E(G) = \{e_1, e_2, ..., e_m\}$ of ordered pairs of nodes called directed arcs or arrows.

Given the directed arc $e_k = (v_i, v_j)$, the connection that exists between the node v_i and the node v_j does not have the same value as the connection that exists

between v_i and v_j; we have therefore that, unlike the non-oriented case, the arc $e_k = (v_i, v_j)$ is distinct from the arc $e_h = (v_j, v_i)$ [11].

The arc $e_k = (v_i, v_j)$ of an oriented graph is characterized by the direction, graphically represented by an arrow exiting from the first node of the pair (v_i) called tail, it is also said that the arc e_k is exiting from v_i, and by an arrow entering in the second node of the pair (v_j) called head. The two nodes v_i and v_j are said to be extremes of the arc e_k, and the arc e_k is 'incident' upon v_i and v_j. Similarly to the non-oriented case, we define the star of v_i in G, denoted by $\omega(v_i)$, as the set of arcs 'incident' upon v_i.

The star $\omega(v)$ can be partitioned into an incoming star $\omega^-(v)$, the set of arcs entering v, and an outgoing star $\omega^+(v)$, the set of arcs leaving v. Induced subgraphs and subgraphs are defined as for the undirected case. The vertex degree v_i, usually denoted by $d_g(v_i)$, is given by the incoming degree (number of arcs $\omega^-(v)$) added to the outgoing degree (number of arcs $\omega^+(v)$). The concepts of isolated and hanging nodes as expressed in the previous paragraph remain unchanged.

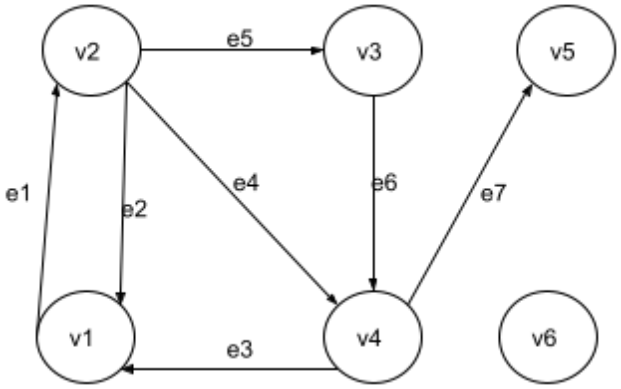


Figure 3.5: Example of directed graph

As before there are some examples of the concepts introduced above in Figure 3.5:

- Arch e_1 and e_2 are distinct.
- e_1 has v_1 as tail and v_2 as head;
- $\omega(v_1)$ is the star of v_1 and it is composed by:$\{e_1, e_2, e_3\}$;
- Incoming star: $\omega^-(v_1) = \{e_2, e_3\}$;
- Outgoing star: $\omega^+(v_1) = \{e_1\}$;

- v_3 has degree equal to 3, given by the outgoing degree (1) and the incoming degree (2);

- v_6 is an isolated node.

In the case of an oriented graph we speak instead of oriented path, given by the sequence of consecutive arcs of the type $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, ..., $e_k = (v_k, v_{k+1})$ with departure from vertex v_1 and arrival in vertex v_{k+1} passing through the intermediate vertices $v_2, v_3, ..., v_k$. The node preceding the arc in the path must be its tail, while the node following the arc must be its head. The presence of such a path makes v_{k+1} reachable from v_1 and introduces the concept of reachability. This aspect is not symmetrical, but represents the oriented extension of the concept of connection seen above.

3.4 Graph representation

In graph theory, it is possible to represent a graph $G = (V, A)$ in various ways, such as the graphical representation with circles for the nodes and lines joining the circles for the arcs, or also the extensive one when listing the vertices and the arcs. The graphical representation can be very immediate and intuitive, but when there are many arcs and nodes, the graphical representation is no longer feasible. For this reason new ways of representing graphs have been adopted, which can be chosen according to the problem to be solved, the size of the graph, etc. There are three possible representations[15]:

1. Adjacency matrix;

2. Incidence matrix;

3. Adjacency list.

The first case is more used in graph problems with a much higher number of arcs than the number of nodes because the representation with the matrix serves to have a more immediate and intuitive understanding.

The second is less efficient from a computational point of view than the others, but is still preferable in some specific cases such as when we have an oriented graph and we only need information about the direction of the arcs.

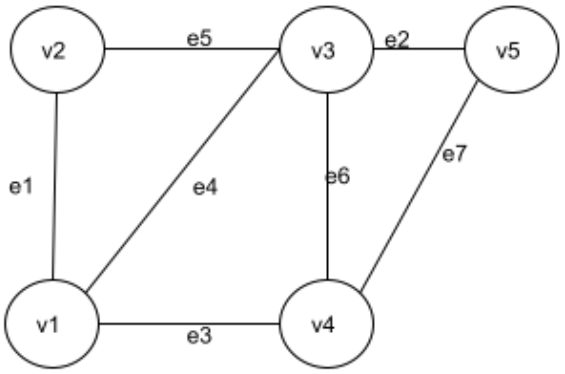
Finally the third, the adjacency list is preferred when the graph under consideration is sparse, i.e. with |E| (number of arcs) similar to |V| (number of vertices).

3.4.1 Adjacency matrix

For the representation with adjacency matrices it is assumed that the vertices are numbered sequentially from 1 to |V|. We represent a graph $G = (V, E)$ with a binary square matrix $Q = (q_{ij})$ of size $|V| \times |V|$ such that[16] [13]: do the staple with the two values for q_{ij}:

$$\begin{cases} q_{ij} = 1 & \text{if } (i,j) \in E \\ q_{ij} = 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The adjacency matrix of an undirected graph is symmetric, i.e. the value q_{ij} is equal to q_{ji}; in this case it is possible to simplify the matrix and consider only the data above the diagonal (including the diagonal), reducing the space needed to store the matrix by half (Figure 3.6).



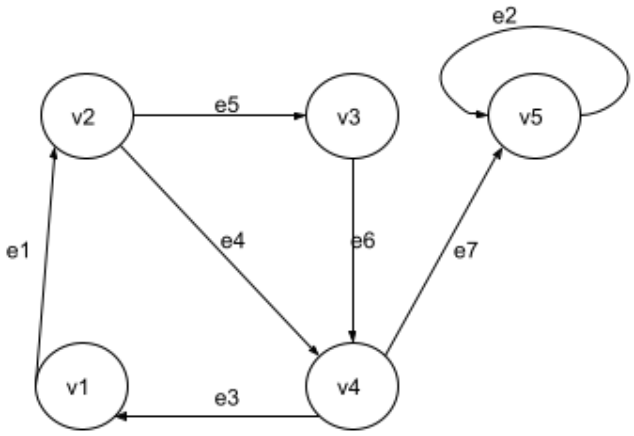
	v1	v2	v3	v4	v5
v1	0	1	1	1	0
v2		0	1	0	0
v3			0	1	1
v4				0	1
v5					0

Figure 3.6: Adjacency matrix undirected graph

In Figure 3.6 there is a visualization of an undirected graph with the adjacency matrix having only half of the values, while in Figure 3.7, the representation of an oriented graph is shown; in this case the matrix is not symmetrical and clearly all the elements inside it must be memorized in order to solve the problem. In the case where the graph is weighted, the values of the weights q_{ij} are defined in this way:

$$\begin{cases} w(i,j) = 0 & \text{if i=j} \\ w(i,j) = w(i,j) & \text{if } i \neq j \text{ and (i,j) belongs to E} \\ w(i,j) = \infty & \text{if } i \neq j \text{ different and does not belong to a} \end{cases} \tag{3.2}$$

The weighted graph is used in problems where, for example, it is necessary to pay attention to the distance or time of the arcs.



	v1	v2	v3	v4	v5
v1	0	1	0	0	0
v2	0	0	1	1	0
v3	1	0	0	1	0
v4	0	0	0	0	1
v5	0	0	0	0	1

Figure 3.7: Adjacency matrix directed graph

In Figure 3.7 there is an example of a visualization of an oriented graph using an adjacency matrix.

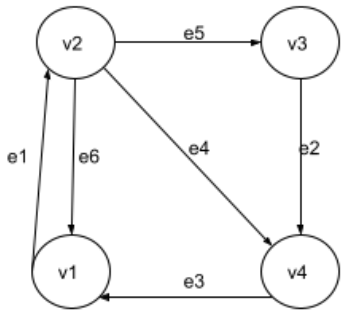
3.4.2 Incidence matrix

For the representation with incidence matrices, the vertices are assumed to be numbered in an arbitrary sequence from 1 to |V|. An undirected graph $G = (V, E)$ is represented by a matrix $Q = (q_{ij})$ of size $|V| \times |E|$ such that:

$$\begin{cases} q_{ij} = +1 & \text{if } e_j \text{ is incident on } v_i \\ q_{ij} = 0 & \text{if } e_j \text{ is not incident on } v_i \end{cases} \tag{3.3}$$

In the case of an oriented graph, the generic element q_{ij} of the incidence matrix must take into account the direction of the arc under examination. An oriented graph $G = (V, E)$ is represented by a matrix Q = (q_{ij}) of dimensions $|V| \times |E|$ such that

$$\begin{cases} q_{ij} = +1 & \text{if } e_j \text{ is an arc entering } v_i \\ q_{ij} = -1 & \text{if } e_j \text{ is an outgoing arc from } v_i \\ q_{ij} = 0 & \text{otherwise} \end{cases} \tag{3.4}$$



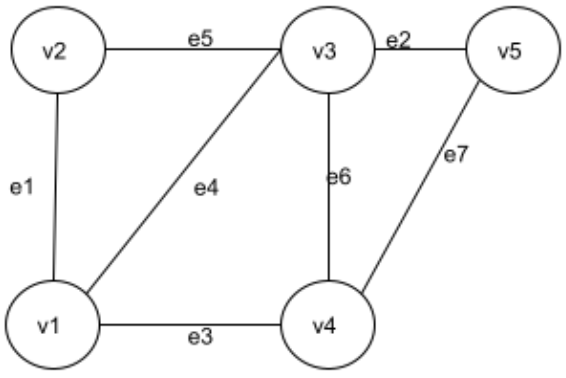
	v1-v2	v2-v1	v2-v3	v2-v4	v3-v4	v4-v1
v1	-1	1	0	0	0	1
v2	1	-1	-1	-1	0	0
v3	0	0	1	0	-1	0
v4	0	0	0	1	1	-1

Figure 3.8: Incidence matrix

This mode of representation, shown in Figure 3.8, is based on an n x m matrix in which each row corresponds to a vertex (node) and each column is associated with an edge (arc). In the incidence matrix, therefore, each column has two elements different from zero, in correspondence to the matrix rows relative to the two extreme nodes of the arc, except when an arc has only one node (loop) as its extremes.

3.4.3 Adjacency list

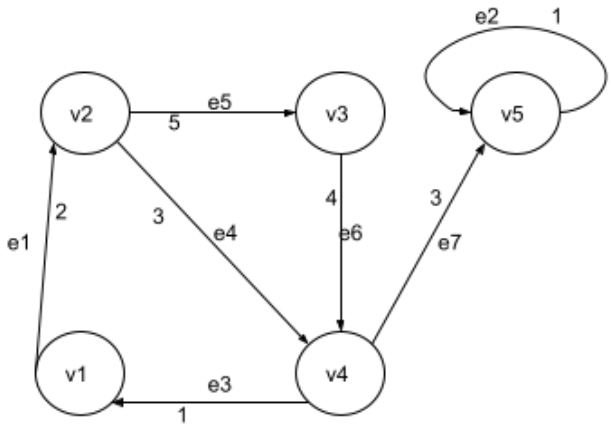
The adjacency list allows efficient storage of graphs, $G = (V, E)$, and is represented by a vector Adj of lists, one list for each vertex of the graph. For each vertex v_i, $Adj[v_i]$ contains all the vertices v_j adjacent to v_i, i.e., all those vertices v_j such that there exists an arc $(v_i, v_j) \in E$, in particular, this set of vertices, according to arbitrary order, is stored as a list.



Node	Adjacency list
v1	v2,v3,v4
v2	v1,v3
v3	v1,v2,v4,v5
v4	v1,v3,v5
v5	v3,v4

Figure 3.9: Adjacency list undirected graph

Figure 3.9 shows the use of adjacency lists to visualize an undirected graph. Here it is possible to observe the fact that for every node there is a list of nodes adjacent to itself, so for example v_1 has v_2, v_3, v_4 as adjacent nodes.



Node	Adjacency list
v1	v2-2
v2	v3-5,v4-3
v3	v4-4
v4	v1-1,v5-3
v5	v5-1

Figure 3.10: Adjacency list directed graph

Instead Figure 3.10 shows the use of adjacency lists of a weighted directed graph. Here it is possible to observe the list of nodes adjacent to itself, and the weights associated to each arc.

Chapter 4

Vehicle routing problems

4.1 VRP definition

Vehicle routing problems are at the centre of distribution management. They have been the subject of intense research for more than 50 years, because of their huge scientific interest for two main reasons:

- Challenging combinatorial optimisation problems;
- Relevance in many practical situations, such as transport, logistics, communications, and so on.

Some of the real cases are the pick-up and haul of municipal waste to regulated landfills, the design of routes for electric vehicles with stops at charging stations, the transportation of people with limited mobility.

The large number of studies and publications shows that the scientific community has made a great effort for more than 50 years in trying to solve these problems. since each case studied may present distinct characteristics from what has already been discussed previously in the literature, there are always new variants of the general problem that continue to appear and require new differentiated studies [].

Many progresses have been made since the first publication of the paper on the truck dispatching problem by Dantzig and Ramser (1959). Strong formulation models have been proposed and numerous heuristics have been developed for this class of problems.

In the case of a heterogeneous fleet of vehicles, which have to provide a service of collection and delivery of products to a known set of customers with respective demands, it is necessary to find and identify short routes that respect all the characteristics of the problem, in an economically efficient way, in order to represent a key force and role in transport logistics. In this sector there is a class of logistics problems known as the Vehicle Routing Problems, commonly abbreviated as VRPs,

whose solutions aim to minimise the total distances travelled by vehicles during the collection and delivery of products to customers[13].

The Vehicle Routing Problems are a set of problems whose objective is to design minimum permissible routes for a fleet of vehicles to serve a certain number of customers under certain constraints, which vary depending on the case examined.

Typically, the problem involves each customer being served and all of them being assigned to vehicles in such a way that the maximum capacity of the vehicle is not exceeded during the journey. The main aspects to be optimised are either:

- The reduction to a minimum number of vehicles used;
- The reduction of the distance travelled consisting of the customer's depot and any other collection points.

Then planning the routes on which customers are to be reached and served with the aim of minimising vehicle use and transport costs. The logistics problem taken into consideration can be described and explained by defining in detail all the characteristics of the problem, which constitute fundamental elements for the resolution of the VRP itself.

The main properties common to this type of problem are warehouses, vehicles, road network and customers, which can be represented graphically through the use of graphs, with the arcs representing the road network and each arc being associated with a weight corresponding to the distance or travel time of the relative arc. Customers and depots are represented by vertices, which are the ends of the arcs in the graph, so we will have two vertices or nodes for each arc.

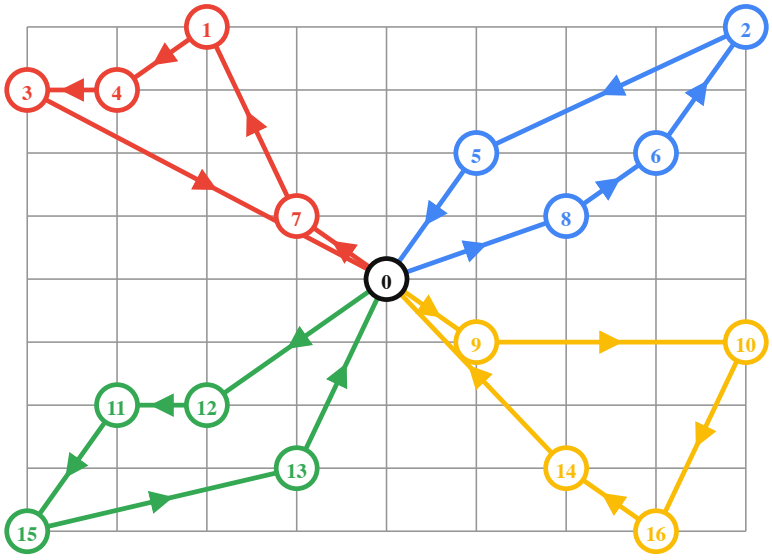


Figure 4.1: VRP example

4.1.1 Depots

The starting point for the study of the problem is the depot which has common or different characteristics depending on the hypotheses considered[13]:

- They are always represented graphically as vertices of the graph and are the starting points for all vehicles;
- They can be multiple or the depot can be unique, sometimes the problem can also be subdivided for each depot;
- Each depot is associated with a certain number and type of vehicles and can be linked to a working time within which the vehicles must leave and return to the assigned depot;
- They are also the point of arrival, as the vehicles must return to their depot for reassignment at the end of their journey.

4.1.2 Vehicles

Each depot is associated with a certain number of vehicles with characteristics determining the classification of VRP problems:

- The fleet of vehicles associated with a depot can be homogeneous or heterogeneous both in terms of size and capacity;
- The maximum vehicle capacity depends on the type of vehicle and the type of saturation considered (this may be by weight, volume or number of pieces transported);
- Each vehicle is associated with a driver for whom the cost of the work performed is calculated (loading/unloading cost, hourly cost, ...);
- Other costs associated with the vehicle are linked to the length of the route assigned in terms of both space and time;
- Each vehicle is assigned a route (the set of routes is a subset of the road network that corresponds to the set comprising all the arcs of the graph).

4.1.3 Customers

The VRP study is structured around the needs of the parties who are to receive the service, i.e. the customers:

- They are represented by the vertices of the graph;

- Each of them is associated with a certain demand which is made explicit as the quantity of goods to be loaded and/or unloaded, which the vehicles (and therefore the drivers) are to be loaded and/or unloaded, which the vehicles (and therefore the drivers) are committed to carry correctly and efficiently to their destination;

- The service must be provided during certain periods of time linked to the opening and closing times of each customer;

- They have different delivery/collection needs depending on the goods handled by each of them (e.g. if it is perishable or not or if it needs to be transported and undergo further need to be transported and further processed by other parties);

- Each customer may be served by a single vehicle which carries out the entire request or the service may be carried out by several vehicles (split deliveries), always depending on the assumptions made;

4.1.4 Properties of the VRPs

Therefore the VRPs implie a service which one or more customers require to be satisfied, according to specific requirements, by vehicles which are located in one or more depots and make their journeys through the "available" road network and according to the specifications of the individual route assigned to them.

The VRPs make it possible to determine a set of trips so as to serve all customers in accordance with the operational constraints considered with a view to minimising the cost function. The main objectives, some of which are conflicting, of VRPs problems are:

- To minimise the number of vehicles used to serve all customers;

- To minimise the total distance travelled by the fleet;

- To minimise the total cost of transport which depends on the total distance travelled, the total time spent and the total cost of transport;

- To minimise the penalties associated with the service being completed to only a part of the customers;

- To balance routes with regard to travel time and/or vehicle load;

- To minimise an objective function that corresponds to a combination of the above objectives.

All parts involved must be subject to some constraints, without which a correct analysis would be impossible. Some of these constraints are common to the different types of VRPs, while others are specific to each type of problem or are the starting point of the study from which it cannot be separated. Some typical constraints are:

- The total demand of customers placed along the route cannot exceed the capacity of the vehicle assigned to it;
- The customers served can only request delivery of goods, only collection or both;
- The customers may only be served in their specific time windows and during the drivers' working periods;
- Any priority constraints defined between customers must be respected. For example, if some of the goods to be delivered to a customer have to be picked up by others first (pickup and delivery problem), or if entire groups of customers have to be served by the same vehicle. Another situation is the so-called VRPs with Backhauls, in which vehicles can pick up and deliver, provided that the latter activity takes place first.

Before moving on to the description of the different variants of a VRPs problem, it should be noted that this type of problem having the execution time through algorithms that search for the exact solution increases exponentially with the size of the problem itself.

4.2 VRPs classification

The extremely broad range of actual applications where routing issues are found leads to the definition of many VRPs variants with additional characteristics and constraints, called attributes, aiming to capture a higher level of system detail or decision choices, including but not limited to richer system structures (e.g., several depots, vehicle fleets, and commodities), customer requirements (e.g., multi-period visits and within-period time windows), vehicle operation rules (e.g., load placement, route restrictions on total distance or time, and driver work rules), and decision context(e.g., traffic congestion and planning over extended time horizons).

These attributes lead to a variety of Multi-Attribute Vehicle Routing Problems (MAVRPs), making up a vast research, development, and literature domain. The dimensions of most problem instances of interest hinders the applicability of exact methods, while the few software systems presented as general heuristic solvers are increasingly challenged by the growing variety of attributes. Finally, some MAVRPs combining multiple attributes together, the so-called rich VRPs, may

be especially difficult to solve because of the compound, and possibly antagonist, decisions they involve. The vehicle routing domain, vast and difficult to classify, has been historically articulated around several streams of research dedicated to a number of major attributes. Such diverse research lines would be justified if the nature of the various problem settings would call for radically different solution approaches. Yet, MAVRPs naturally share many common features, and most heuristic strategies developed for specific problems can be applied to a broader range of VRPs variants. We identify, and classify, 14 notable MAVRPs, which have been the object of a consistent body of well-acknowledged research resulting in a considerable number of heuristic methods and a number of common benchmark sets of test instances [17].

4.2.1 Capacitated (CVRP)

The simplest VRPs are the Capacitated Vehicle Routing Problem (CVRP), which assumes an identical fleet of vehicles, with uniform finite capacity, located in one central depot in order to service a set of customers with related demands. The objective is to determine a set of vehicle trips with minimum total distance/time cost, having as condition that each vehicle starts and ends at the depot, each client is visited exactly once, and the total demand of the route cannot exceed the total capacity.

4.2.2 Multiple depots (MDVRP)

The multi-depot VRPs consist in a number of depots greater than 1, with the property that each vehicle is assigned to one depot. As in the other cases the depot is both the origin and the destination of the vehicle's route.

4.2.3 Backhauls (VRPB)

The vehicle routing problems with backhauls has two types of customers: linehaul customers (deliveries), and backhaul customers (pick-ups). In any route there is a mix of both groups, and the linehaul customers must precede all the backhaul customers. For this reason, routes consisting of backhaul customers only are not admitted.

4.2.4 Time Windows (VRPTW)

The Vehicle Routing Problems with Time Windows are a vehicle routing problem in which a fleet of identical vehicles of finite capacity and located at a central depot must service a set of customers with known demands and specific time window.

The time window associated with customer i is denoted by (e_i, l_i); the service of each customer must necessarily start at a time t_i contained within the time window. In the case of early arrival at customer i, the vehicle must wait for the earliest time (e_i) before being able to carry out the service, generating a waiting cost; in the opposite case, if the vehicle arrives after the latest time (l_i), the solution is not feasible. Hence it is allowed to wait for an early arrival to a customer but a late arrival is not allowed.

A service time is associated with each customer, indicating the time interval during which the vehicle carrying out the service remains at the customer's premises. The objective is to find a set of vehicle routes having minimum total cost (distance or time), with each vehicle starting and ending at the depot, each client visiting exactly once and within his/her time window, and the total demand handled by any vehicle does not exceed its capacity.

4.2.5 Distance-Constrainted (DVRP)

The Distance-Constrainted Vehicle Routing Problem has a constraint on the maximum distance length, or maximum time, of the route to be covered. In particular, each arc is associated with a non-negative length, and the total length, given by the sum of the arcs considered, cannot exceed the maximum length imposed as a constraint. If the vehicles are different from each other, then the maximum values considered are different for each of them. If the parameter associated to the arc represents the travel time, each node i has assigned a service time s_i which represents the time needed for the vehicle to perform its service at the customer; Hence the time at every node i will increase of travelling time of arc x_{ji} with $t_{ji} + s_i$.

4.2.6 Profits (VRPP)

A maximization problem of the total profit with the possibility to not visit all nodes. The goal is to visit the nodes which maximize the sum of collected profits while respecting a vehicle time limit. Vehicles, as always, are required to start and end at the depot.

4.2.7 Pickup and Delivery (VRPPD)

The pick-up and delivery problem is a problem with the aim of finding a set of optimal routes with minimum distance, for a fleet of vehicles, that serve a certain set of demands. all vehicles have a given capacity, a start position and a final position. the transport demands are characterised by a load that needs to be transported, an origin and a destination. In conclusion, the pick-up and delivery problem is about constructing optimal routes to visit all pick-up and delivery locations and

satisfying precedence and coupling constraints. Precedence constraints concern the limitation in which each pick-up location must be visited before visiting the respective delivery location. Pairing constraints limit the set of acceptable routes in order that a vehicle has to do both pick-up and delivery of the load of a request[18].

4.2.8 LIFO

«Similar to the VRPPD, but in this case another restriction is placed on the loading of the vehicles: at any delivery location, the item being delivered must be the item most recently picked up. This scheme reduces the loading and unloading times at delivery locations because there is no need to temporarily unload items other than the ones that should be dropped off.»[19]

4.2.9 FIFO

«The FIFO VRP model requires that if a vehicle leaves customer i to go to customer j at any time t, any identical vehicle with the same destination leaving customer i at a time $t + \varepsilon$, where $\varepsilon > 0$, will always arrive later. This is an intuitive and desirable property though it is not present in all models.»[18]

4.2.10 Time-Dependent (TDVRP)

An interesting common extension of the Time Windows and FIFO VRP in urban environments is the Time-Dependent, where the arc costs on the graph depend on the departure date. This situation is an actual example of most cities since the time taken to travel from a location to another one depends in most of the cases on the traffic load, which varies during the day [20].

4.2.11 Dynamic (DVRP)

Another extension to standard VRP is called the Dynamic Vehicle Routing Problem, and his main characteristics of this type of problem are the uncertainty in the data and it can be due to different sources, and it can have different natures. It is when the service requests are not completely known before the start of service, but they arrive during the distribution process. Since new orders arrive dynamically, the routes have to be replanned at run time in order to include them.

4.2.12 Period (PVRP)

«The period vehicle routing problem is a variation of the classic vehicle routing problem in which one has to assign deliveries (or pickups) to customer locations

in a manner that each location is visited at the required frequency during the planning horizon (usually one week), and the cost of the delivery (or pickup) routes is minimized. All deliveries (or pickups) are from a single depot.»[18]

4.2.13 Open (OVRP)

«Distribution network design is a crucial problem that companies care about much more than the past. Open VRP is a variant of VRP in which each route is a sequence of customers with a deterministic demand and predefined geographical location, that starts at depot and ends at one of the customers (in contrast to classical VRP where it returns to the depot after servicing period). Practically, OVRP is applied when suppliers does not have a vehicle fleet and prefer outsource their transportation and distribution affairs. One should be noted that it is much more economical for suppliers to outsource the distribution of the goods or materials. In the OVRP, it is supposed that each customer is visited once by a single vehicle and the total demand of all customers allocated to a vehicle (route) does not exceed the vehicle capacity and the objective function is usually minimizing the overall traveling cost.»[21]

4.2.14 Stochastic (SVRP)

Customer demand in many cases is known before the routes need to be planned because orders have been received at the central depot. However, in some situations, the size of the demand from a customer may be unknown until the vehicle arrives at the customer's place. This is an example of a stochastic vehicle routing problem where routes must be planned based on the probability distribution of the demand at any customer. In such problems, a strategy needs to be defined that explains what happens when a vehicle runs out of the commodity it is carrying before it has completed all its deliveries, for example, returning to the depot by the shortest route in order to reload[18].

Chapter 5

Heuristic techniques for solving a VRP

5.1 Introduction

The term Heuristics indicates an algorithmic method for finding admissible (not necessarily optimal) solutions of an optimisation problem. For this class of problems, the development of heuristic techniques with the aim of finding a good admissible solution, close to the optimum and obtained in short processing times, is of fundamental importance.

5.2 Classification of heuristic techniques

The construction of effective heuristic algorithms requires a careful analysis of the problem to be solved in order to identify the "structure", i.e. the specific useful characteristics, and a good knowledge of the main algorithmic techniques available. In fact, even though each problem has its own specific characteristics, there are general techniques that can be applied, in different ways, to many problems, producing well-defined classes of optimisation algorithms, as set out below.

5.2.1 Constructive heuristics

Constructive heuristic algorithms are oriented towards the identification (construction) of an admissible solution, found by implementing one path at a time through a series of steps and containing the cost during the process. They can be schematised into the following three steps[13]:

1. Initialisation: first of all the path is started by looking for a starting element for the construction of the partial solution S;
2. Selection of a new element to be added to the partial solution: the selection criterion for a new element to be added to the partial solution S is identified;
3. Stopping criterion: if S is complete, and therefore admissible, the procedure is stopped, otherwise we return to the previous step.

Then, constructive heuristic algorithms can be divided into two categories:

- Sequential: they construct one route at a time until all vertices are exhausted; one cannot choose between several routes in which to insert a vertex;
- Parallel: they construct several routes at once. The number of routes can be fixed at the beginning or derive from the progressive merging of smaller routes already computed. proposed routes are optimised through the exchange of vertices between adjacent clusters.

Savings Heuristics

This is probably the best known heuristic algorithm proposed for VRP, and it applies naturally to problems for which the number of vehicles is not predefined. This procedure, proposed for the first time by Clarke and Wright (1964), starts with n distinct routes in each of which the customer is served by a dedicated vehicle; at each iteration the objective is to aggregate two or more nodes together, but also two or more routes with more than one customer, and calculate the saving resulting from these operations. Given two customers, each served by a dedicated vehicle with a distinct route, the application of saving highlights the potential savings that can be achieved by using a single vehicle to visit first one customer and then the next, travelling a single route. it can be performed in two versions: parallel and sequential.

- Common operations:
  For $i, j = 1, ..., n, i \neq$j are calculated saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. n routes of the type (0, i, 0) are created for $i = 1, ..., n$, and the savings are sorted in descending order;
- Sequential version:
  Each generic route (0, i, . . . , j, 0) is considered in rotation, and the first saving s_{ki} or s_{jl} is determined that allows it to be merged with another route containing the arc (k, 0) or the arc (0, l) to give rise to a new admissible route. If this step is successful the new route is created with the merge, otherwise this step is always applied to the next route. The algorithm stops when no more merging is possible;

- Parallel version:
  It is considered the ordered savings and proceed by determining whether it is possible to merge together two existing routes containing the arc (0, j) and the arc (i, 0) respectively, given the save s_{ij}, resulting in a new admissible route.

Tested on some known instances of the problem, this algorithm offers in its parallel version the best performance, although the results remain far from the known optimal solutions. Both versions also produce good routes initially but then tend to lose interest, sometimes becoming too geographically extensive. To overcome this defect it has been proposed to use a route shape parameter, λ, that modifies the saving according to the formula: $s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$. In this way more emphasis is given to the distance of the vertices to be connected, for values $\lambda \geq 1$.

Insertion Heuristics

The insertion algorithms are considered as constructive-sequential algorithms, the initialisation phase, the choice of the customer to be inserted in the route as the first node, is carried out following one of the three following criteria [13]:

1. Customer with the greatest distance from the depot;
2. Customer with a shorter closing time (of its own time window);
3. Customer with the lowest cost in terms of time and distance. from the depot.

After initialising a new route, two criteria, $c_1(i, u, j)$ and $c_2(i, u, j)$, are used to insert at each iteration a customer u within the current route between two adjacent nodes i and j. Let $(i_0, i_1, i_2, ..., i_m)$ be the route being considered, with $i_0 = i_m = 0$. For each client u that is not yet assigned, and therefore out of the route, the best fit to the route under consideration is calculated: $c_1(i(u), u, j(u)) = \min c_1(i_{p-1}, u, i_p)$ with $p = 1, ..., m$ By inserting client u between i_{p-1} and i_p, all service start times for nodes in the route $(i_0, i_1, i_2, ..., i_m)$ can be altered. The next client to be inserted will be the one for which the following relation applies: $(c_2(i(u), u, j(u)) = \text{optimum } (c_2(i(u), u, j(u)))$ where u is without route and feasible. Client u^* is then inserted into the route between $i(u^*)$ and $j(u^*)$. This iteration continues until it is no longer possible to find customers that respect the constraints and therefore the considered route ends; if not all customers have been satisfied yet, a new route is initialised. Three different specific approaches are now considered, based on the general criterion just presented[13].

- Insertion Heuristic I1:
  The first criterion seeks to maximise the benefit of serving a customer present in the partial route being constructed rather than having to serve the customer

with a direct route.
$c_{11}(i,u,j) = d_{iu} + d_{uj} - \mu d_{ij}$ with $\mu \geq 0$
$c_{12}(i,u,j) = e_{ju} - e_j$
The variable e_{ju} indicates the new instant at which the service at customer j can begin since node u has been added to the route. $c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j)$
with $\alpha_1 + \alpha_2 = 1$ and with $\alpha_1, \alpha_2 \geq 0$
$c_2(i,u,j) = \lambda d_{0u} - c_1(i,u,j)$ with $\lambda \geq 0$
This type of method maximises the benefit of inserting the customer within the partial route, rather than served by direct routing. The best insertion is the one that minimises a weighted combination of distance and time, i.e. minimises a measure of extra distance and extra time required to visit a customer. Clearly different values of μ and λ lead to different criteria for selecting the insertion of a node in the route;

- Insertion Heuristic I2:
  The second type of criterion selects the customers to be inserted in the route with the objective of minimising costs both in terms of time and total distance.
  $c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j)$
  with $\alpha_1 + \alpha_2 = 1$ and with $\alpha_1, \alpha_2 \geq 0$
  $c_2(i,u,j) = \beta_1 R_d(u) - \beta_2 R_t(u)$
  with $\beta_1 + \beta_2 = 1$ and with $\beta_1, \beta_2 \geq 0$
  $R_d(u)$ and $R_t(u)$ denote the total distance and total time of the partial trip that are generated with customer u inserted, respectively;

- Insertion Heuristic I3:
  In the third approach the urgency of serving a customer is also taken into account within the time aspect.
  $c_{11}(i,u,j) = d_{iu} + d_{uj} - \mu d_{ij}$ with $\mu \geq 0$
  $c_{12}(i,u,j) = e_{ju} - e_j$
  $c_{13}(i,u,j) = l_u - e_u$
  $c_1(i,u,j) = \alpha_1 c_{11}(i,u,j) + \alpha_2 c_{12}(i,u,j) + \alpha_3 c_{13}(i,u,j)$ with $\alpha_1 + \alpha_2 + \alpha_3 = 1$
  $c_2(i,u,j) = c_1(i,u,j)$
  with $\alpha_1, \alpha_2, \alpha_3 \geq 0$.

In all the approaches presented the insertion of a customer in a route is guided and conditioned by both geographical and temporal criteria. It is therefore expected that the waiting time in the solutions proposed by this heuristic algorithm will be significantly lower than that proposed by the distance-only criteria.

5.2.2 Two-stage heuristics

Two-stage methods decompose the problem into the operations of subdividing vertices into groups (clusters) and constructing admissible routes. In turn, two-stage algorithms can be divided into two classes[22]:

- Cluster first - Route second:
  The vertices are initially grouped into clusters and a route is then constructed for each cluster;
- Route first - Cluster second:
  A route is constructed on all vertices and then subdivided.

Cluster-First, Route-Second: Sweep Algorithm

In this section a will explain an example of Cluster-First, Route-Second. The diffusion of this algorithm is attributed to Gillett and Miller for solving planar VRP instances. The idea is to group the vertices in sets, clusters, according to their angular position with respect to the repository, and then solve a TSP instance for each cluster. Some implementations foresee that at the end of the procedure the proposed routes are optimised by exchanging vertices between adjacent clusters. We assume that each vertex i is represented, with respect to the repository, with its polar coordinates (θ_i, ρ_i), where θ_i represents the angle and ρ_i the distance from the repository. The angle θ_i is calculated with respect to the value of reference θ_i^* relative to an arbitrary vertex i^* . The steps of the algorithm are as follows:

1. The vertices are numbered according to increasing values of θ_i;
2. A free vehicle k is selected;
3. Starting from the free vertex with the minimum value of θ_i, we assign vertices progressively to vehicle k until the capacity constraint of the same is not violated. Eventually, at each insertion it is possible to optimise with local search procedure. If at the end of this step there are still free vertices, the execution is resumed from point 2;
4. For each cluster defined in this way, a TSP instance is resolved, in an exact or approximate way.

Route-First, Cluster-Second: Cluster-Second

These algorithms foresee the preliminary construction of a tour as a solution of a TSP instance on all the vertices of the graph under consideration, ignoring any constraints of the VRP, such as, for example, the capacity of the single vehicle.

In a second moment the tour is decomposed into more routes considering the constraints of the problem: these routes represent the final solution to the VRP instance.

It has been shown that the second step of route partitioning is equivalent to solving a minimum path instance on an acyclic graph, and therefore can very well be done using, for example, Dijkstra's algorithm in $O^{(n^2)}$ time. Although theoretically very simple as methods, in practice no relevant results of these algorithms are recorded on VRP benchmarks.

5.2.3 Improved heuristics

Improvement heuristics are applied to a pre-existing solution (in some cases even ineligible) with the intention of improving it and typically operate by swapping arcs or vertices between different routes.

Ejection chain

The ejection chain method was devised by Glover, and is a widely used heuristic algorithm for improving solutions to routing problems. This method bases its operation on a series of "state changes" of the elements that make up the solution of the problem. The changes cause these elements to undergo an ejection from their current state to a new state to be determined. An ejection chain is initiated by selecting a set of elements from the current problem solution; these elements are modified in such a way that a change of state can be performed, taking them from their current configuration to a new configuration. The result of the state change performed on these elements leads to the eventual identification of other groups of elements on which to apply the same state changes performed previously. Thus, when an ejection chain is initiated, the elements of a solution undergo state changes and ejections from their current state in an alternating way, often triggering a domino effect. The ejection chain may or may not end. In the first case, the chain ends when, after changing the configuration of some elements of the solution, it is no longer necessary to eject other elements from their current state. If applied to the vertices of a single path, the ejection chain method can be useful to decrease the number of vehicles used in a given solution of a routing problem: in fact, if the ejection chains are initialized with the vertices of a given path and if these chains all have a termination, the path will eventually be empty and can, therefore, be eliminated from the solution of the problem[23].

2-opt

2-opt is one of the best known and most widely used local search algorithms for routing problems. The proximity set defined by the operator involves examining all

the solutions that can be obtained, starting from the initial one, with an exchange between any pair of non-adjacent arcs within a given path. Swapping a pair of arcs within a freight delivery trip means identifying and removing this pair, thus breaking the route into two distinct paths; subsequently, it is necessary to reconnect these paths by creating two new arcs and reversing the order in which customers visit between the exchanged arcs. The purpose of the 2-opt operator is to decrease the total distance covered by the route on which it is applied, thus reducing the value of the total global distance of the solution[23].

Relocate 1-0

The relocate 1-0 transfers a single vertex from one path to another by inserting it between two consecutive vertices; the purpose of applying this method is to decrease the total distance travelled. It is possible to have other variants as Relocate 1-1, Relocate 2-1, and so on. In the case of 1-1 the exchange is of a couple of vertices in two different paths, with always the goal to minimize total distance or time.

Chapter 6

Vehicle Routing Problem of a set of small food shops in a local area of Torino

6.1 LINKS Foundation

"The LINKS Foundation – Leading Innovation & Knowledge for Society (LINKS) is a non-profit private Foundation founded in 2016 with the aim to boost the interaction between research and the business world towards the internationalization of the local socio-economic system. LINKS promotes, conducts, and strengthens innovation and research projects processes, improving products implementation as well as the study of new approaches and models. LINKS is the result of the merger of two distinguished research and innovation institutions: ISMB (Istituto superiore Mario Boella) founded in 2000, focusing on ICT, and SiTI (Istituto Superiore sui Sistemi Territoriali per l'Innovazione) founded in 2002, focusing on Territorial Systems and Smart Cities. Similarly to ISMB and SITI, this Foundation has been founded by Compagnia di San Paolo and Politecnico di Torino.

LINKS relies on technological and process competences of around 160 researchers working in close cooperation with companies, academia and Public Authorities. The Foundation is involved in several European, national and regional funded projects, participates in international Bodies and maintains an open dialogue with the local, national and international business entrepreneurship, in order to share and enhance knowledge, experience and innovation. LINKS is involved in several industrial cooperation activities with both large enterprises and SMEs, as well as in various higher-education initiatives in partnership with academic institutions.

LINKS extends its activities to facilitate innovation, employing its technical

excellences and results, into domains that are interdisciplinary and aligned with the priority themes of the European agenda, such as Energy, Mobility, Industry and Circular Economy.

The "Urban Mobility & Logistic Systems" (UML) research area supports private enterprises and Public Authorities to bring innovation and sustainability in the fields of people mobility, freight transport and logistics, through a multidisciplinary approach. LINKS-UML research is focused on:

- Transport planning (Regional Transportation Plans, Urban Plans for Sustainable Mobility, Public Transport Plans);
- Transport and logistics modelling and optimization (predictive models for mobility, traffic models, optimization of logistic systems);
- User's behaviour, users' engagement and innovation acceptance/adoption;
- Estimation of transport impacts.

LINKS-UML is composed of transport and environmental engineers, mathematicians, and spatial planners and its main research projects are related to the domains of Mobility (public transport & MaaS, shared mobility, cycling and walking, electric mobility, connected & autonomous mobility, urban air mobility) and Logistics (multimodal freight transport, last-mile & city logistics, data exchange processes)."[24]

6.1.1 Urban Mobility & Logistic Systems

The Urban Mobility & Logistic Systems area supports companies and public administration in bringing innovation and sustainability to the areas of personal mobility and merchandise transport through a multidisciplinary approach.

Composed of transport and environmental engineers, mathematicians and territorial planners, the team is able to follow the entire process of applying innovation to transport: from analyses and specialised studies, to support for the definition of interventions and policies in a participatory manner, to the modelling and simulation of scenarios, through to support for the experimentation of innovative practices and technologies and the assessment of impacts.

At international level, the area is present in the Advisory Board of the EGTC Interregional Alliance for the Rhine-Alpine Corridor, in ERTICO (network of ITS actors) and in EIP-SCC (Marketplace of the European Innovation Partnership on Smart Cities and Communities).

The area collaborates with the Polytechnic of Torino within ICELab (Laboratory of ICT technologies for the integrated and intelligent management of logistics and business in urban areas), with the City of Torino within the Torino City Lab

ecosystem (aimed at supporting the experimentation of innovative practices and technologies in the city) and supports the planning activities of the Transport Department of the Piedmont Region, in the framework of the Memorandum of Understanding signed between the Region, Confindustria Piemonte, Politecnico di Torino and Compagnia di San Paolo. The main scientific areas covered:

- Transport planning (e.g. Regional Transport Plans, Sustainable Urban Mobility Plans);
- Public transport, multimodality and MaaS;
- Predictive mobility and traffic models;
- Optimisation of logistics systems;
- ITS (Intelligent Transport Systems);
- Sustainable, electric, shared and autonomous mobility;
- Mobility behaviour, equity and accessibility;
- Estimation of transport impacts.

In the logistics sector, as an example, UML has collaborated with the Piemonte Region for the preparation of technical contributions for the development of freight transport and regional logistics, useful for the preparation of the future Regional Logistics Plan, collaborates with Confesercenti on the BORGO SAN PAOLO +SMART study, in which is included the activity of defining the functional and technological requirements of a last mile logistics management model in a sustainable way, and is involved in the Interreg ITA-CH Typicalp project, aimed at increasing competitiveness and sustainability of small and medium enterprises in the dairy supply chain in the mountain areas of Valle d'Aosta (IT) and Vallese (CH) through collaborative logistics and supply chain monitoring using blockchain.

6.1.2 Collaboration LINKS - Confesercenti of Torino and Province, Borgo San Paolo + Smart project

The world of commerce is undergoing rapid change, and has seen and is still experiencing radical changes in terms of both supply (from the expanding role of large-scale distribution, to the expansion of franchising and the e-commerce and marketplace "revolution") and demand (changes in priorities and consumption patterns, the shift from purchasing to other forms of availability of goods, as recognised by the so-called sharing economy).

Proximity trade is penalised in competition with other forms of trade because of the small size of enterprises, which makes it difficult to sustain investments in

infrastructure, new technologies and digital skills. Micro and small enterprises in the sector therefore struggle to seize the opportunities that the digital economy could offer them.

The project that is the subject of this offer, "Borgo San Paolo + Smart", will represent the start of a process that will see a sample of micro and small enterprises in the trade, services and tourism sectors at the centre of a series of initiatives for technology transfer, innovation and digital skills enhancement, given that this type of enterprise is normally on the margins of these processes. The service offered has a twofold purpose:

- Firstly, it aims to identify and test technological solutions that can be used by the sample of selected enterprises in order to improve their activities in the area;
- Secondly, it intends to propose possible physical transformations of some points of the district under study and to identify places that can host new shared functions to support local commerce, thus elaborating specific proposals for the requalification and improvement of the urban context to be submitted to the public decision-maker.

The project that is the subject of this offer therefore aims to support, in various ways, neighbourhood commerce enterprises in order to:

1. Use new technologies to achieve greater competitiveness in the management of specific aspects of their business (logistics, home deliveries, order management, communication and marketing, promotion, opening hours, etc.);
2. Rethink some public spaces, making them more attractive and usable by citizens living in and visiting the area, using innovative technological tools and possibly experimenting with new types of business;
3. Plan new ways of using the empty spaces on the street level, sharing possible proposals to be presented to the competent Administrations.

6.2 Formulation of the problem

The project that is the subject of this thesis aims to support, in various ways, collaborative commerce enterprises through the use of optimization technologies to achieve greater competitiveness in the management of specific aspects of business such as logistics, home deliveries, orders management etc.

The problem is defined on a graph $G = (V, A)$, in which the set of vertices $V = P \cup D \cup \{0\}$. $P = \{1, ..., p\}$ is the set of pick up nodes, and $D = \{p+1, ..., p+n\}$ is the set of delivery nodes where $|D| = n$ and $p \leq n$. The starting and ending

depot is defined by the node 0 in G. Let $R = \{r_1, ..., r_m\}$ be the set of requests to be routed where $|R| = m$ and $m \geq n$, every customer can make an order to one or multiple stores, thus each customer can make multiple requests and each shop is assigned to multiple requests from multiple customers. Each request $r \in R$ is represented by one pick up node $p_r \in P$ and one delivery node $d_r \in$D with a volume quantity q_r. All the pick up nodes must be visited before the delivery nodes for each request $r \in R$. K is the set of identical capacitated vehicles that can be used $|K| = p$ located at the depot 0. The set of arcs is $A = V \times V$, each arc $(i, j) \in A$ has an associated travel time $t_{ij} \geq 0$. It is assumed that the travel times satisfy the triangular inequality: $t_{ij} \leq t_{il} + t_{lj}$ $\forall i, j, l \in V$. Each node $n_i \in N$ has a service time s_i for loading or unloading at that node n_i. Each node $p_i \in P$ has a quantity of requests to load onto the vehicle $Q_i \geq 0$, which is the sum of quantities of all the requests having node i as the pick up node. The sum of the volumes loaded on the vehicle is constrained by the capacity of the vehicle C_k. All the orders loaded on the vehicles must be delivered in the T_k hours slot time, so every vehicle $k \in K$ can do a tour of at most T_k. The objective is to minimise the total distance travelled by the vans and also to minimise the number of vans used.

The actual situation, given that no coordination is made, is that every retailer carries out the deliveries for their customers with a distinct (privately owned) vehicle. This problem belongs to the broader class of *Pickup&Delivery* problems (PDP) [25]. The focus here is on a variant that allows for multiple visits to locations. Specifically, the routing problem can have multiple location visits as resulting from divisible pickups and deliveries. Other variants of the classic problem may include single commodity problems with split loads, that is, everything that is to be picked up from (delivered to) a location has the same destination (origin) (e.g. [26]). For divisible pickups and deliveries, each location can serve as a pickup and/or delivery point for multiple commodities (e.g. [27]). That is, every location may require transportation of loads to and/or from multiple other locations. Our problem classifies as a problem with divisible pickups and deliveries. Moreover the problem is restricted to a maximum length for each route as a result of a working shift for drivers and capacitated vehicles. Very recent papers on similar variants with single and multiple vehicles of PDP are [28], [29] and [30].

6.3 Mathematical model

In this section I propose the mathematical model formulated with the following variables:

- X_{ijk} equals to 1 if arc (i, j) is traversed by vehicle k, 0 otherwise;
- Y_k equals to 1 if vehicle k is activated, 0 otherwise;

- T_{ik} indicating the time of vehicle route at node i with vehicle k;
- L_{ik} indicating the load volume of the vehicle at node i with vehicle k.

This problem can be formulated as follows:

Obj. Function

$$Min \sum_{i \in V} \sum_{j \in V, i \neq j} \sum_{k \in K} (X_{ijk}(t_{ij} + s)) + (C_{auto} - s) \sum_{k \in K} Y_k \tag{6.1}$$

Constraints

$$\sum_{j \in (V-0)} X_{0jk} \leq Y_k \quad \forall k \in K \tag{6.2}$$

$$\sum_{i \in V-j} \sum_{k \in K} X_{ijk} = 1 \quad \forall j \in P \tag{6.3}$$

$$\sum_{i \in V-j} X_{ijk} = \sum_{l \in V-j} X_{jlk} \quad \forall k \in K \quad \forall j \in V \tag{6.4}$$

$$T_{ik} + t_{ij} + s_i - T_{jk} \leq M(1 - X_{ijk}) \quad \forall i \in V \quad \forall j \in V - i \quad \forall k \in K \tag{6.5}$$

$$(-s) + \sum_{i \in V} \sum_{j \in V-i} X_{ijk}(t_{ij} + s) \leq T_{Max-route} \quad \forall k \in K \tag{6.6}$$

$$L_{ik} - L_{jk} - \sum_{b \in P} \sum_{a \in V-b} (X_{abk} q_{bj}) \leq M(1 - X_{ijk}) \quad \forall j \in D \quad \forall i \in V \quad \forall k \in K \tag{6.7}$$

$$L_{ik} + Quant_j - L_{jk} \leq M(1 - X_{ijk}) \quad \forall i \in V \quad \forall j \in P \quad \forall k \in K \tag{6.8}$$

$$L_{ik} \leq C_k \quad \forall k \in K \quad \forall i \in V \tag{6.9}$$

$$T_{ik} \leq T_{jk} \quad \forall (i,j) \in R \quad \forall k \in K \tag{6.10}$$

$$\sum_{a \in V-j} X_{ajk} \geq \sum_{l \in V-i} X_{lik} \quad \forall (i,j) \in R \quad \forall k \in K \tag{6.11}$$

$$T_{0k} = 0 \quad \forall k \in K \tag{6.12}$$

$$L_{0k} = 0 \quad \forall k \in K \tag{6.13}$$

$$X_{ijk} \in \{0,1\} \quad \forall i \in V \quad \forall j \in V \quad \forall k \in K \tag{6.14}$$

$$X_k \in \{0,1\} \quad \forall k \in K \tag{6.15}$$

$$L_{ik} \geq 0 \quad \forall i \in V \quad \forall k \in K \tag{6.16}$$

$$T_{ik} \geq 0 \quad \forall i \in V \quad \forall k \in K \tag{6.17}$$

- Equation 6.1 is the objective function which minimize the route time and the number of vehicle used. s is the service time at each delivery pick up node, so at each route the service time is counted as the number of arcs minus 1 (so if the vehicle is used or not);
- Equation 6.2 is the constraint to activate the vehicle k if it departs from depot;
- Equation 6.3 is the constraint in order to visit once each pick up node;
- Equation 6.4 is used to respect the flow entering in a node equals to the flow exiting from the same node;
- Equation 6.5 It is needed to update route time at each node during the path in each route, and also it eliminates sub tours;
- Equation 6.6 is the constraint to respect the maximum route time;
- Equation 6.7 It is needed to update load volume on the vehicle at each delivery node during the path in each route;
- Equation 6.8 It is needed to update load volume on the vehicle at each pick up node during the path in each route;
- Equation 6.9 is the constraint to respect the maximum load capacity of the vehicle;
- Equation 6.10 is the constraint to ensure that for each request the vehicle passes from the pick up node before to passing from the delivery node;
- Equation 6.11 is the constraint it to ensure that if the vehicle passes from one pick up node it has to passes from all the delivery nodes which have a request starting from that pick up node;

- Equation 6.12 is set the initial value of route time at 0;
- Equation 6.13 is set the initial value of volume capacity at 0.

6.4 Initial heuristic

In this section there is the explanation of the first heuristic version used to solve the problem. Now there is a high level description of the algorithm and then more detailed one.

Algorithm 1 Heuristic Algorithm

```
1: procedure CREATION ROUTES
2:     for each shop do
3:         TSP(Input:Nodes,Output:Path)
4:         Add Path in Best
5:         Add Path in Bench
6:     Calculate Centroids(Input: Nodes; Output: Distance Matrix Centroids)
7:     for c in C do
8:         Merge Relaxed Routes(Input: Bench, c, Distance Matrix Centroids;
   Output: Elite Routes)
9:         for each route in Elite Routes do
10:            Reconstruct Route(Input: route; Output: Route)
11:            Improved Route(Input: Route; Output: Improved Route)
12:            Add Improved Route in Sol
13:        if All Improved Route are feasible AND Objective value ≤ Best then
14:            Best = Sol
15:            Sol = 0
16:    Return Best
```

6.4.1 TSP, N routes for N shops

Having N shops, I solve the tsp mathematical model for each one finding the minimum route starting from the shop and finally returning to the shop. It starts by creating as many routes as there are shops, i.e. the case where each one only delivers to its own customers. In order to find the route for each shop I make the TSP which allows me to find the minimum route starting from the shop, delivering all the products and returning to the same shop at the end of the route. In this case I use these results as benchmark for the algorithm's solution. Then I will unify the routes in sets in order to minimize total route and number of vehicles used,

having one node more as the common depot where all vehicles start and end their routes.

6.4.2 Centroid for each route

Considering that each shop can have different area of their customers I wanted to implement the fact to consider the centroid of each initial route in this way:

Algorithm 2 Calculate the centroid coordinates for each route and the distance matrix

procedure CENTROID
Input 1: A: set of N sets of vertices for each routes
Output: $DistanceMatrix$: distance matrix for all the nodes
B is an empty set
while A not empty **do**
▷ Extract all vertices of one set in A and calculate two coordinates
$x_j = (\sum_{i=0}^{n} x_i)/(n+1)$
$y_j = (\sum_{i=0}^{n} y_i)/(n+1)$
Insert (x_j, y_j) in B ▷ it is related to the j route
for k in B **do**
for l in B **do**
$dist(k,l) = ((x_k - x_l)^2 + (y_k - y_l)^2)^{0.5}$
Add $dist(k,l)$ in $DistanceMatrix$
Return $DistanceMatrix$

6.4.3 Merge Relaxed Routes

In this section a will explain how I decided to merge the routes in few groups. At the beginning I decided to find more solutions and then pick the best one among the feasible solutions through the relaxation of the constraints using factors C from 1 to 2, which multiply the capacity and time constraints. The factors can be changed, increasing or decreasing the number of factors for each sets according to the user. For the resolution of the 1350 instances I decided to pick only these values for the two sets in order to have fast execution time:

- Capacity factors=[2,1.5,1.2,1.1,1];
- Times factors=[2,1.5,1.25,1.11,1]

With these two sets I used all the combinations of pairs of values from them, obtaining 25 initial states:

- (2,2);
- (2,1.5);
- (2,1.25);
- (2,1.11);
- (2,1);
- (1.5,2);
- (1.5,1.5);
- (1.5,1.25);
- (1.5,1.11);
- (1.5,1);
- (1.2,2);
- (1.2,1.5);
- (1.2,1.25);
- (1.2,1.11);
- (1.2,1);
- (1.1,2);
- (1.1,1.5);
- (1.1,1.25);
- (1.1,1.11);
- (1.1,1);
- (1,2);
- (1,1.5);
- (1,1.25);
- (1,1.11);
- (1,1);

These are all the starting states used for the merge algorithm, it is necessary to include the pair (1,1) because in case every shop route has almost used the capacity and time constraints the solution of this merge probably will be the same as the initial solution. This is a limit case that can occur.

Merge knapsack1

The Merge knapsack1 is used to simply minimize the number of vehicles used with the constraints of capacity and time of the pair explained above. It finds the minimum number of vehicles to use and cluster them in groups.

Formulation of the problem:
In this problem there are $N = \{1, ..., n\}$ shops and at most $M = \{1, ..., m\}$ vehicles with M=N. Each shop is related to his route hence it has a capacity q_i which is the total load amount at the beginning of the route. Each shop has also a time constant related to the time for his single route t_i. Every shop has to be assigned to 1 vehicle. Every vehicle can have at most s_n shops with $1 \leq s_n \leq n$ which satisfy the constraints of time, at most T hours (T is defined in the previous paragraph and it has values $T \geq 360$) as the sum of each route assigned to the vehicle. Then also it has to respect the capacity constraint, at most C dm^3 (C is defined in the same way as T $C \geq 1$)as the sum of each route assigned to the vehicle.
Mathematical model of Knapsack1:

Obj. Function

$$Min \sum_{j=1}^{m} Y_j \tag{6.18}$$

Constraints

$$\sum_{j=1}^{m} X_{ij} = 1 \quad \forall i \in N \tag{6.19}$$

$$\sum_{i=1}^{n} X_{ij} q_i \leq C \times Y_j \quad \forall j \in M \tag{6.20}$$

$$\sum_{i=1}^{n} X_{ij} t_i \leq T \times Y_j \quad \forall j \in M \tag{6.21}$$

$$Y_{j-1} \geq Y_j \quad \forall j \in N - 1 \tag{6.22}$$

$$Y_j \in Z^+ \quad \forall j \in N \tag{6.23}$$

$$X_{ij} \in \{0,1\} \quad \forall j \in N \quad \forall i \in M \tag{6.24}$$

- Equation 6.18 is the objective function needed to have the minimum number of vehicles;

- Equation 6.19 is the constraint in order to have all shops assigned to one vehicle;
- Equation 6.20 is the constraint in order to respect capacity constraint, hence the sum of the capacity of all the shops assigned to one vehicle must be lower or equal to C (max capacity) if the vehicle is used;
- Equation 6.21 is used to respect time constraint, hence the sum of the time of all the shops assigned to one vehicle must be lower or equal to T (max time) if the vehicle is used;
- Equation 6.22 is a constraint to reduce symmetry, which allowed to use the variables Y_j in order so firstly Y_1 is activated then Y_2 and so on.

I run this model with a maximum time of 10 seconds to find the solution, always to have a solution with low execution time. As result I will have the minimum number of vehicles used, and this number is used for the Merge knapsack2.

Merge knapsack2

The Merge knapsack2 is used to have the shops with centroid coordinates nearest to the shops contained in each vehicles with a the number of vehicles found with Merge knapsack1. In this way it is easier and faster to find the minimum number of vehicles to use and cluster them in sets.

Formulation of the problem:
In this problem there are N={1,...,n} shops and at most M={1,...,m} vehicles with m $\leq$ n, the m number is found with Merge knapsack1. Each shop is always related to his route hence it has a capacity q_i which is the total load amount at the beginning of the route. Each shop has also a time constant related to the time for his single route t_i. Every shop has to be assigned to 1 vehicle. Every vehicle can have at most s_n shops with $1 \leq s_n \leq n$ which satisfy the constraints of time, at most T hours (T is defined in the previous paragraph and it has values $T \geq 360$) as the sum of each route assigned to the vehicle. Then also it has to respect the capacity constraint, at most C dm^3 (C is defined in the same way as T $C \geq 1$)as the sum of each route assigned to the vehicle. Furthermore, in this model I added the distances between the centroids D_{ij}.

Mathematical model of Knapsack2:

Obj. Function

$$Min \quad M + \sum_{i=1}^{m} \sum_{j=1}^{m} Z_{ij} D_{ij} \tag{6.25}$$

Constraints

$$\sum_{j=1}^{m} X_{ij} = 1 \quad \forall i \in N \tag{6.26}$$

$$\sum_{i=1}^{n} X_{ij} q_i \leq C \times Y_j \quad \forall j \in M \tag{6.27}$$

$$\sum_{i=1}^{n} X_{ij} t_i \leq T \times Y_j \quad \forall j \in M \tag{6.28}$$

$$Y_{j-1} \geq Y_j \quad \forall j \in N - 1 \tag{6.29}$$

$$X_{ij} + X_{i_2 j} \leq Z_{ii_2} + 1 \quad \forall j \in N \quad \forall i \in M \quad \forall i2 \in M - i \tag{6.30}$$

$$Y_j \in Z^+ \quad \forall j \in N \tag{6.31}$$

$$X_{ij} \in \{0,1\} \quad \forall j \in N \quad \forall i \in M \tag{6.32}$$

$$Z_{ii_2} \in \{0,1\} \quad \forall i_2 \in M \quad \forall i \in M \tag{6.33}$$

- Equation 6.25 is the objective function needed to have the minimum number of vehicles;
- Equation 6.26 is the constraint in order to have all shops assigned to one vehicle;
- Equation 6.27 is the constraint in order to respect capacity constraint, hence the sum of the capacity of all the shops assigned to one vehicle must be lower or equal to C (max capacity) if the vehicle is used;
- Equation 6.28 is used to respect time constraint, hence the sum of the time of all the shops assigned to one vehicle must be lower or equal to T (max time) if the vehicle is used;
- Equation 6.29 is a constraint to reduce symmetry, which allowed to use the variables Y_j in order so firstly Y_1 is activated then Y_2 and so on;
- Equation 6.30 is a constraint to activate Z_{ii_2} variable if i and i_2 are in the same vehicle.

This time, I run this model with a maximum time of 2 seconds to find the solution, always to have a solution in a small amount of time. As result I will have the shops assigned to each vehicle with the minimum distance between the shops of each vehicle. The final results of this section are the *eliteroutes*.

6.4.4 Reconstruct Route

After having assigned the shops with their relative customers to the vehicles, I start creating the route for each *eliteroutes*. In this procedure I wanted to create different solutions and picking the best 15 results to optimize them with another Relocate 1-0 Constraints.

Nearest Neighbour

Nearest Neighbour is used to create the initial routes.

Algorithm 3 Nearest Neighbour

procedure CREATE ROUTE WITH NEAREST NEIGHBOUR METHOD
    Input 1: P: set of nodes
    Input 2: $Constraints$: set of fixed nodes of the final route
    Input 3: $DistMatrix$: distance matrix for all the nodes
    Output: Sol: route's duration and route
    Sol $\leftarrow Constraints$
    for each node in 1,..,$|P - Constraints|$ **do**
        From last node in Sol find the shortest edge between all nodes in $P - Sol$
        Insert the node in Sol
    Return Sol

Improved Route

Improved Route through the Relocate 1-0 Constraints improves the first 15 routes of each vehicle.

Algorithm 4 Improved Route

```
    procedure OPTIMISE THE PATH OF A VEHICLE
        Input 1: Route: path to optimize of one vehicle
        Input 2: DistMatrix: distance matrix for all the nodes
 4:     Input 3: Shops
        Output: ImprovedRoute: best route
        while Route is improved or number of improvements < 300 do
            for each node i in 1,...|Route| do
 8:             if i is a shop then:
                    for each node j in i,...|Route| do
                        if j is not a shop then
                            break
12:                     else
                            Place i after j
                            if the route is improved then
                                Save Route and break
16:                         else
                                Replace i to the original position
                else
                    for each node j in i,...|Route| do
20:                     Place i after j
                        if the route is improved then
                            Save Route and break
                        else
24:                         Replace i to the original position
        ImprovedRoute ← Route
        Return ImprovedRoute
```

Reconstruct Route

In Reconstruct Route there is the procedure to create the routes for each combination of the initial states.

Algorithm 5 Reconstruct Route

procedure ROUTES CREATION
Input 1: $Veic$: shops assigned to vehicles
Input 2: $Cust$: customers for each shop
Input 3: $DistMatrix$: distance matrix for all the nodes
Input 4: $Shops$
Output: $Routes$: all routes for each vehicle
Ba is an empty list ▷ routes' combinations
P is an empty list ▷ route's nodes
$VeicRoutes$ is an empty list ▷ routes' nodes for each vehicle
for each set of stores in $Veic$ **do**
if number of permutation of set of stores > 500 **then**
Insert a list of 500 random combinations of the stores in Ba
else
Insert a list of all permutation of the stores in Ba
for each set of stores in $Veic$ **do**:
for each combination in Ba **do**
Constraints $\leftarrow (depot, firststore)$
for each shop in $|combination|$ **do**
if shop is the first one **then**
Insert shop's unique customers in P
Insert the depot and the next shop in P
Sol $\leftarrow NearestNeighbour(P, Constraints, DistMatrix)$ ▷ Sol is the solution of Nearest Neighbour, (time, route)
Add to $Constraints$ all the nodes until the next shop in Sol
else if shop is not the last one **then**
Insert shop's unique customers in P
Insert the next shop in P
Sol $\leftarrow NearestNeighbour(P, Constraints, DistMatrix)$
Add to $Constraints$ all the nodes until the next shop in Sol
else
Insert the remaining customers in P
Sol $\leftarrow NearestNeighbour(P, Constraints, DistMatrix)$
Insert Sol in $VeicRoutes$
for each $Veic$ **do**
for Best 15 routes in $VeicRoutes$ **do**
Do $ImprovedRoute(Route, DistMatrix, Shops)$
Insert the best Route among the results in $Routes$
Return $Routes$

6.4.5 Selection of the best routes

For all the initial states we create the routes and at the end we select the routes with minimum number of vehicle and minimum travel time in this way:

Algorithm 6 Select the best routes

```
    procedure SELECT THE BEST ROUTES
        Input 1: Route: path to optimize of one vehicle
        Input 2: DistMatrix: distance matrix for all the nodes
        Input 3: Shops
        Output: BestRoutes: best routes
 6:     for each combination of T, C do
            Feasible ← True
            VeicP ← Mergeknapsack1(DistMatrix, Route)
            Veic ← Mergeknapsack2(Route, DistMatrix, |VeicP|)
            if Veic is 0 then Veic ← VeicP
            Sol ← ReconstructRoute(Veic, Cust, DistMatrix, Shops)
12:         for each vehicle in Sol do
                if Route time> 360 then
                    Feasible ← False
                    break
            if Feasible = True then
                for each vehicle in Sol do
18:                 if Capacity of any nodes is not respected then
                        Feasible ← False
                        break
            if Feasible = True then
                Insert Sol in BestRoutes
        Keep in BestRoutes all the routes with minimum number of vehicles
24:     Keep in BestRoutes the routes with minimum total route time
        Return BestRoutes
```

6.5 Final Heuristic

The final algorithm is really similar to the first one, it changes from Creation routes: I pick the best 8 results and other 10 between the 9^{th} and the last one, because if I pick only the first results I have routes that are already optimized and from the depot the vehicle passes by a lot of stores without passing by customers and it could not respect the capacity constraint. Moreover I changed also the Relocate 1-0 Constraints in order to check the capacity constraint for each improvement.

Final Improved Route

Algorithm 7 Final Improved Route

```
procedure Optimise the path of a vehicle
    Input 1: Route: path to optimize of one vehicle
    Input 2: DistMatrix: distance matrix for all the nodes
    Input 3: Shops
    Input 4: Orders capacities associated to each order
    Input 5: Capacity: vehicle capacity
    Output: BestRoute: best route
    for each node in Route do
        Calculate vehicle capacity at the node.
    if at any node the vehicle capacity > Capacity then
        Return 0
    while Route is improved or number of improvements < 300 do
        for each node i in 1,...|Route| do
            if i is a shop then:
                for each node j in i,...|Route| do
                    if j is not a shop then
                        break
                    else
                        Place i after j
                        if (the route is improved) and (vehicle capacity at any node ≤ Capacity) then
                            Save Route and break
                        else
                            Replace i to the original position
            else
                for each node j in i,...|Route| do
                    Place i after j
                    if (the route is improved) and (vehicle capacity at any node ≤ Capacity) then
                        Save Route and break
                    else
                        Replace i to the original position
    BestRoute ← Route
    Return BestRoute
```

Final Reconstruct Route

Algorithm 8 Final Reconstruct Route

```
1: procedure FINAL RECONSTRUCT ROUTE
2:     Input 1: Veic: shops assigned to vehicles
3:     Input 2: Cust: customers for each shop
4:     Input 3: DistMatrix: distance matrix for all the nodes
5:     Input 4: Shops
6:     Input 5: Capacity: vehicle capacity
7:     Input 6: Orders: capacities associated to each order
8:     Output: Routes: all routes for each vehicle
9:     Ba is an empty list                          ▷ routes' combinations
10:    P is an empty list                           ▷ route's nodes
11:    VeicRoutes is an empty list                  ▷ routes' nodes for each vehicle
12:    for each set of stores in Veic do
13:        if number of permutation of set of stores > 500 then
14:            Insert a list of 500 random combinations of the stores in Ba
15:        else
16:            Insert a list of all permutation of the stores in Ba
17:    for each set of stores in Veic do:
18:        for each combination in Ba do
19:            Constraints ← (depot, firststore)
20:            for each shop in |combination| do
21:                if shop is the first one then
22:                    Insert shop's unique customers in P
23:                    Insert the depot and the next shop in P
24:                    Sol ← NearestNeighbour(P, Constraints, DistMatrix)   ▷
   Sol is the solution of Nearest Neighbour, (time, route)
25:                    Add to Constraints all the nodes until the next shop in Sol
26:                else if shop is not the last one then
27:                    Insert shop's unique customers in P
28:                    Insert the next shop in P
29:                    Sol ← NearestNeighbour(P, Constraints, DistMatrix)
30:                    Add to Constraints all the nodes until the next shop in Sol
31:                else
32:                    Insert the remaining customers in P
33:                    Sol ← NearestNeighbour(P, Constraints, DistMatrix)
34:                    Insert Sol in VeicRoutes
```

35: **if** $|VeicRoutes| \leq 18$ **then**
36: **for** each route in $VeicRoutes$ **do**
37: $FinalImprovedRoute(Route, DistMatrix, Shops, Orders, Capacity)$
38: Insert the best Route among the results in $Routes$
39: **if** Route time of best route > 360 **then**
40: $Routes \leftarrow 0$
41: Return 0
42: **else**
43: **for** Best 8 routes in $VeicRoutes$ **do**
44: $FinalImprovedRoute(Route, DistMatrix, Shops, Orders, Capacity)$
45: $Step \leftarrow \frac{(|VeicRoutes|-9)}{10}$
46: **for** routes with step **do**
47: $FinalImprovedRoute(Route, DistMatrix, Shops, Orders, Capacity)$
48: Insert the best Route among the results in $Routes$
49: **if** Route time of best route > 360 **then**
50: $Routes \leftarrow 0$
51: Return 0
52: Return $Routes$

6.5.1 Selection of the best routes

Algorithm 9 Final Selection of the best routes

procedure FINAL SELECTION OF THE BEST ROUTES
Input 1: $Route$: path to optimize of one vehicle
Input 2: $DistMatrix$: distance matrix for all the nodes
Input 3: $Shops$
Output: $BestRoutes$: best routes
for each combination of T, C **do**
$Feasible \leftarrow True$
$VeicP \leftarrow Mergeknapsack1(DistMatrix, Route)$
$Veic \leftarrow Mergeknapsack2(Route, DistMatrix, |VeicP|)$
if $Veic$ is 0 **then** $Veic \leftarrow VeicP$
$Sol \leftarrow FinalReconstructRoute(Veic, Cust, DistMatrix, Shops)$
if $Sol = 0$ **then**
$Feasible \leftarrow False$
else
Insert Sol in $BestRoutes$
Keep in $BestRoutes$ all the routes with minimum number of vehicles
Keep in $BestRoutes$ the routes with minimum total route time
Return $BestRoutes$

Chapter 7

Data analysis

7.1 Instances

The instances used to test the algorithm have a maximum route time of 6 hours per vehicle. The service time is 3 minutes for pick up or delivery service at each node and each request order has a volume between 27 dm^3 and 33 dm^3. After having developed the algorithm I created 1350 different instances in order to test efficiency and analyze the results. Obviously I tried to simulate some possible scenarios related to the Links problem. Therefore, I considered an area comparable to that of a neighbourhood in Turin, but also I tried increasing the area and analysed how it varied.

I classified the instances with these properties:

- Number of nodes in the map: I chose 5 different quantities of nodes corresponding to the shops and the customers and 1 depot:
  - 25 nodes, and 1 depot;
  - 50 nodes, and 1 depot;
  - 100 nodes, and 1 depot;
  - 150 nodes, and 1 depot;
  - 200 nodes, and 1 depot.
- Map size: I used 3 square maps of dimension l x l measured in min^2 as I considered time rather than distance:
  - 15 x 15 min^2;
  - 30 x 30 min^2;
  - 45 x 45 min^2.

- Percentage of shops on the total number of nodes: I analysed 2 different scenarios:
  - 10 percent of the total nodes are shops;
  - 20 percent of the total nodes are shops.
- Repetition of orders: there are 3 cases where the customers can have more than one order to different shops:
  - 0%, in this case every customer has only one order;
  - 15%, in this case there is a 15 percent increase in orders compared to the 0% scenario;
  - 30%, in this case there is a 30 percent increase in orders compared to the 0% scenario.
- Vehicle capacity: I used 3 types of vehicles:
  - 625 dm^3 city car;
  - 1250 dm^3 Fiorino;
  - 2500 dm^3 Doblo.

I combined all these properties and for each I conducted 5 different simulations, amounting to 1350 instances. When creating instances, I simulated randomly the position of nodes, the assigned orders and the quantity per order with relative constraints:

- For the position of the nodes I assumed that the depot can have (x,y) coordinates between 0.3 and 0.7 times the map size, in order to be near the center of the area. On the other hand the location of customers and shops in the map are completely random;
- After having all the coordinates I assigned random the customers to their respective shops;
- The quantity per order is between 27 dm^3 and 33 dm^3 which is a considerable amount of items.

```
26 nodes, 15 min, 10% Shops, 0% Repeated orders, 625 dm^3, Iteration 1
[8.579128117900524, 7.049540169587319]
[11.385109595654363, 9.31341578500704]
[4.734639026145529, 10.583171362710026]
[0.509623046278968, 12.02560020730717]
[3.78342791556957, 11.276520990001945]
[6.370816288284338, 12.845414286164585]
[1.8747466338590009, 7.2753979074027235]
[11.197257813714987, 12.320061218962312]
[3.2329447971263665, 2.896598134994858]
[7.921346630760208, 3.1082980864240186]
[7.388005251067616, 4.852391339912765]
[6.531481966634774, 10.784945019480784]
[1.88896292927565, 13.870281878214858]
[6.663889358626749, 14.626792583411554]
[6.168284105856363, 4.48467836104823]
[9.145793590957314, 0.6116774570241329]
[0.3847534322006646, 14.89390720883822]
[10.99687019810004, 5.367745638769728]
[2.242654677043425, 6.500411710778212]
[11.942122789321662, 4.632144404431776]
[5.5446086178068015, 7.450588062255567]
[8.43999444175042, 11.518786112439637]
[14.317619375842321, 8.034928209113385]
[12.521005533763809, 11.621142174630588]
[8.126464247136461, 14.103652463044794]
[0.4436976385350322, 9.011440629982399]
depot, shops, customers
[0]
[1, 2, 3]
[4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
24, 25]
Customer of each shop, Orders,Total quantities in each shop, Quantity shop-
customer, having 0% percentage repeated orders
[[4, 5, 6, 7, 8, 9, 10, 11], [12, 13, 14, 15, 16, 17, 18, 19], [20, 21, 22,
23, 24, 25]]
[[1, 4, 32], [1, 5, 32], [1, 6, 29], [1, 7, 32], [1, 8, 30], [1, 9, 28], [1,
10, 31], [1, 11, 29], [2, 12, 32], [2, 13, 28], [2, 14, 30], [2, 15, 31],
[2, 16, 32], [2, 17, 28], [2, 18, 31], [2, 19, 30], [3, 20, 31], [3, 21,
29], [3, 22, 31], [3, 23, 29], [3, 24, 31], [3, 25, 27]]
[243, 242, 178]
[[0.0, 243.0, 0.0, 0.0, 32.0, 32.0, 29.0, 32.0, 30.0, 28.0, 31.0, 29.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0,
242.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 32.0, 28.0, 30.0, 31.0,
32.0, 28.0, 31.0, 30.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
178.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 31.0, 29.0, 31.0, 29.0, 31.0, 27.0]]
Dimension of vehicle
625
```

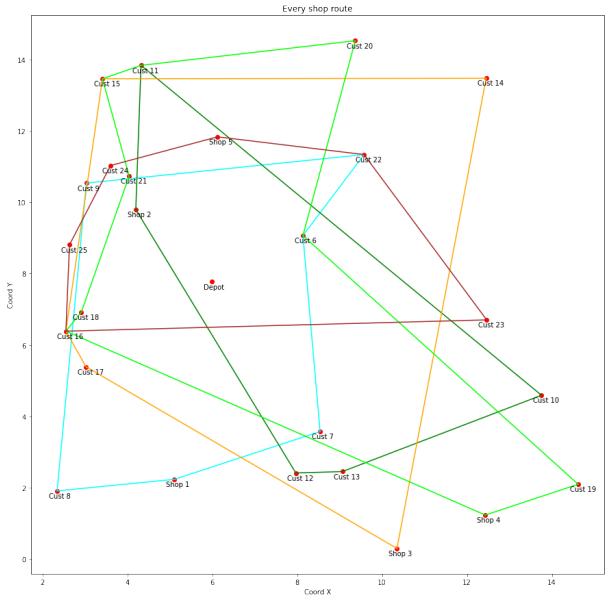
Figure 7.1: Instance example

in Figure 7.1 there is an example of the data for each instance:

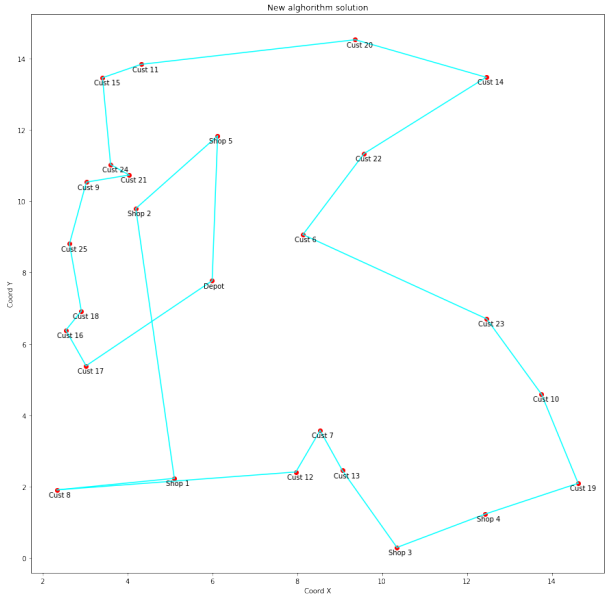
- Coordinates of each node;
- Identification of which nodes are shops, customers, and depot;
- Quantity and customer-shop pairings for each order;
- Vehicle capacity.

Example maps

Here are 3 examples of the algorithm's results in comparison with the initial solutions, cases with 25 nodes and 50 nodes.



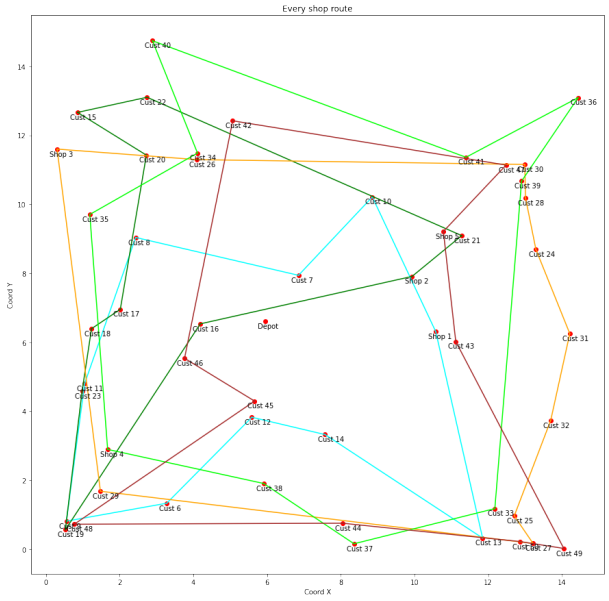
(a) Route for each store: 209 Min, 5 Vehicles



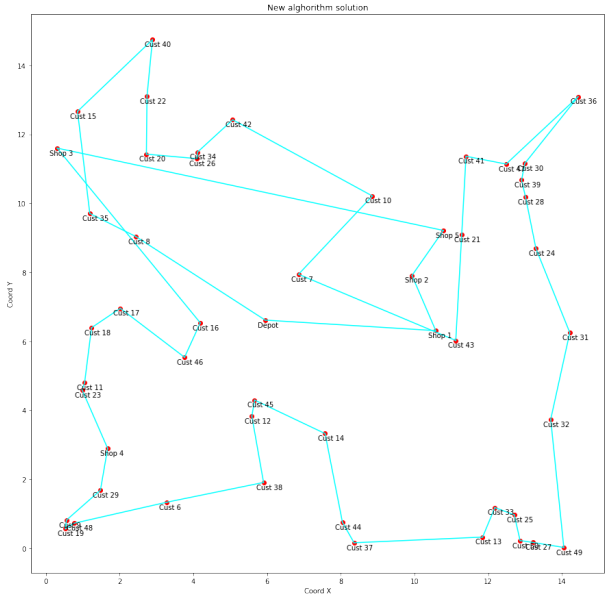
(b) Merged routes: 149 Min, 1 Vehicle

Figure 7.2: 25 nodes, 15 min^2, 5 shops, 1250 dm^3 cap, 0 rep

In Figure 7.2 there is an example of a map, with 20 customers, 5 stores, and 1 depot. The first map shows the initial state while the second one displays the problem solution after the use of the algorithm, going from 5 vehicles to 1 vehicle and from a total time of 209 minutes to 149 minutes.



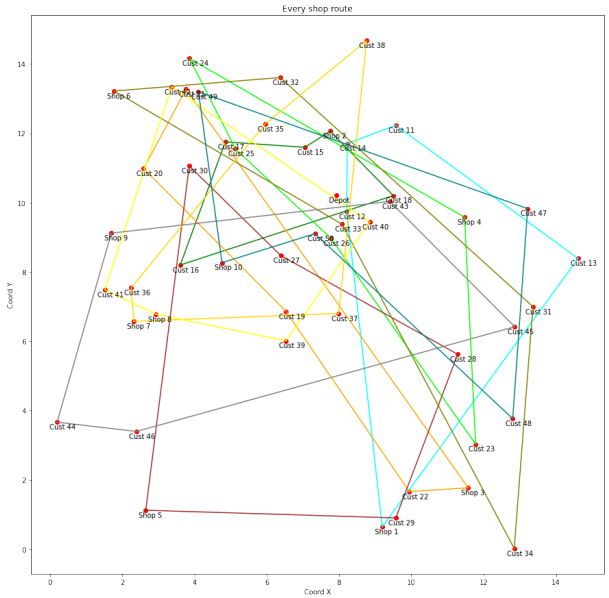
(a) Route for each store: 359 Min, 5 Vehicles



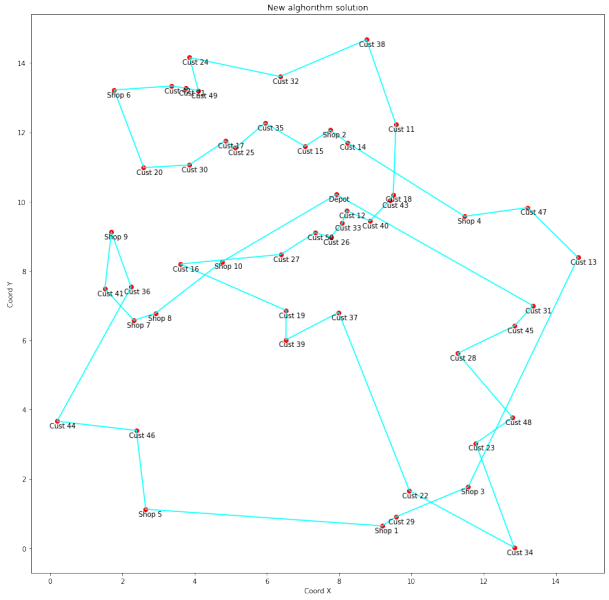
(b) Merged routes: 259 Min, 1 Vehicle

Figure 7.3: 50 nodes, 15 min^2, 5 shops, 2500 dm^3 cap, 0 rep

In Figure 7.3 there is an example of a map, with 50 nodes and 5 shops. As before, the first map shows the initial state while the second one displays the problem solution, going from 5 vehicles to 1 vehicle and from a total time of 359 minutes to 259 minutes.



(a) Route for each store: 403 Min, 10 Vehicles



(b) Merged routes: 254 Min, 1 Vehicle

Figure 7.4: 50 nodes, 15 min^2, 10 shops, 1250 dm^3 cap, 0 rep

In Figure 7.4 there is another example of a map, with 50 nodes and 10 shops. The first map shows the initial state while the second one displays the problem solution after the use of the algorithm, going from 10 vehicles to 1 vehicle and from a total time of 403 minutes to 254 minutes.

7.2 Initial Version vs Final Version

I compared the 2 versions in order to analyse some relevant categories shown on Figure 7.5. The final version improved the results in all the KPI for each category, moreover the execution time was also reduced.

In conclusion there is an improvement of the results and also of the execution time. We could also add other values of capacity and time constraints, further improving the results and having more comparable execution time.

Nodes, % stores	Average shops per car initial alg	Average shops per car final alg	Average % decrease in total time initial alg	Average % decrease in total time final alg	Average execution time initial alg	Average execution time final alg
50,10	2.65	2.97	21.34	23.25	2.11	2.56
50,20	5.56	5.92	38.9	39.34	4.65	5.14
100,10	2.62	3.03	19.02	21.73	7.51	7.98
100,20	5.93	6.22	40.7	40.95	75.62	63.94
150,10	2.56	3.05	16.58	19.62	22.29	20.27
150,20	5.93	6.21	39.8	40.11	102.91	85.67
200,10	2.64	3.07	17.17	19.75	79.48	69.09
200,20	5.92	6.19	39.74	40.06	124.53	102.09

Figure 7.5: Initial Algorithm versus Final Algorithm

7.3 Analysis

The main features that I wanted to examine in more detail are the following extreme values for each instance category:

- Percentage of stores compared to all nodes: 10 or 20 percent;
- Percentage of repeated orders: 0 or 30 percent. Therefore the case in which every customer has only one order and the case in which there is a greater number of repeated orders;
- Type of vehicle used: small (625 dm^3) or big (2500 dm^3).

I used these classifications in order to take into account the extreme cases of all instances and look at relevant variations. Moreover, I wanted to analyze and compare these features in two macro categories:

- The first one classifying the sets according to the number of nodes in the map and considering the cases with 50, 100, 150, 200 nodes. This choice is due to proportionality while still having a relevant number of nodes, with the aim of analyzing the solutions as the nodes increase in order to validate the scalability of the algorithm;
- The second classification was made by map size for 15, 30 and 45 min^2 in order to consider how much the map size affects the algorithm used.

In the first category for each number of nodes there are as many results as the combinations of the features listed above:

1. (10, 0, 625, 15);
2. (10, 0, 625, 30);
3. (10, 0, 625, 45);
4. (10, 0, 2500, 15);
5. (10, 0, 2500, 30);
6. (10, 0, 2500, 45);
7. (10, 30, 625, 15);
8. (10, 30, 625, 30);
9. (10, 30, 625, 45);
10. (10, 30, 2500, 15);
11. (10, 30, 2500, 30);
12. (10, 30, 2500, 45);
13. (20, 0, 625, 15);
14. (20, 0, 625, 30);
15. (20, 0, 625, 45);
16. (20, 0, 2500, 15);
17. (20, 0, 2500, 30);
18. (20, 0, 2500, 45);
19. (20, 30, 625, 15);

20. (20, 30, 625, 30);
21. (20, 30, 625, 45);
22. (20, 30, 2500, 15);
23. (20, 30, 2500, 30);
24. (20, 30, 2500, 45).

For the second category, these are the combinations for each map size:

1. (10, 0, 625, 50);
2. (10, 0, 625, 100);
3. (10, 0, 625, 150);
4. (10, 0, 625, 200);
5. (10, 0, 2500, 50);
6. (10, 0, 2500, 100);
7. (10, 0, 2500, 150);
8. (10, 0, 2500, 200);
9. (10, 30, 625, 50);
10. (10, 30, 625, 100);
11. (10, 30, 625, 150);
12. (10, 30, 625, 200);
13. (10, 30, 2500, 50);
14. (10, 30, 2500, 10);
15. (10, 30, 2500, 15);
16. (10, 30, 2500, 20);
17. (20, 0, 625, 50);
18. (20, 0, 625, 100);
19. (20, 0, 625, 150);

20. (20, 0, 625, 200);
21. (20, 0, 2500, 50);
22. (20, 0, 2500, 100);
23. (20, 0, 2500, 150);
24. (20, 0, 2500, 200);
25. (20, 30, 625, 50);
26. (20, 30, 625, 100);
27. (20, 30, 625, 150);
28. (20, 30, 625, 200);
29. (20, 30, 2500, 50);
30. (20, 30, 2500, 100);
31. (20, 30, 2500, 150);
32. (20, 30, 2500, 200).

7.3.1 Categories: 50-100-150-200

Category: 50

All the following tables will have average values for each instance since there were 5 simulations for each one of them. In the case of 50 nodes with 10 percent of stores it is possible to observe from Figure 7.6 that the lowest values of number of stores per vehicle are corresponding to map size 45 min^2 and 30 percent repeated orders (9,12). This is because in very large maps the distances are very high and since the customers are random points in the map and not located close to each other, 6 hours of time are not enough to satisfy the customers of many stores. Instead in the case of 15 min^2 map size with lower distances it is possible to merge more stores with one vehicle.

In addition, with 30 percent of repeated orders it is obvious that this ratio (shops per vehicle) is less because there are more orders to be met. Also with 45 min^2 map size it can not join many stores with a vehicle in common. For this reason it will not take advantage of the fact of making a single delivery for two or more orders (one customer can have more than one orders with 30 % of repeated orders). This ratio is closely correlated with the percentage of the decrease of the route time compared to the initial solution. This is due to joining fewer vehicles

together with low decrease in the total time, in fact both for vehicle capacity 625 dm^3 and 2500 dm^3 the result is no relevant.

Types	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,15)	5	23.58	1.23
2. (10,0,625,30)	3.12	25.01	0.9
3. (10,0,625,45)	2.5	19.65	0.77
4. (10,0,2500,15)	5	28.26	7.72
5. (10,0,2500,30)	5	37.12	2.68
6. (10,0,2500,45)	2.5	20.5	0.88
7. (10,30,625,15)	2.5	18.28	1.17
8. (10,30,625,30)	2.5	22.51	0.96
9. (10,30,625,45)	1.79	11.92	0.78
10. (10,30,2500,15)	5	38.27	6.88
11. (10,30,2500,30)	2.78	26.53	2.9
12. (10,30,2500,45)	1.79	11.92	0.97
13. (20,0,625,15)	10	36.07	6.86
14. (20,0,625,30)	5	34.59	4.47
15. (20,0,625,45)	5	39.91	2.83
16. (20,0,2500,15)	10	36.85	10.33
17. (20,0,2500,30)	5	34.59	5.25
18. (20,0,2500,45)	5	39.89	2.82
19. (20,30,625,15)	10	43.47	5.28
20. (20,30,625,30)	5	37.9	4.07
21. (20,30,625,45)	4.55	41.64	2.87
22. (20,30,2500,15)	10	44.69	8.79
23. (20,30,2500,30)	5	36.89	4.4
24. (20,30,2500,45)	4.55	41.33	2.89

Figure 7.6: Results of 50 nodes

The highest values, 5 stores per vehicle, are in correspondence of 15 min^2 map size and capacity 625 dm^3 or 2500 dm^3 (1,4), 30 min^2 map size and capacity 2500 dm^3 without repeated orders (5) or 15 min^2 map size and capacity 2500 dm^3 with 30 percent of repeated orders (10). In the first cases where there are both 625 dm^3 and 2500 dm^3 there are optimal results due to the fact that not having repeated orders it does not violate the capacity constraint. Therefore there is no need to have a larger vehicle and that with a 15 min min^2 map there is no violation of the time constraint being a more limited area. In the case with 2500 dm^3 capacity and 30 min^2 without repeated orders it is optimal because the time constraint is

respected thanks to the fact that it can load everything from the beginning without considering the capacity constraint since there is a very large vehicle capacity. Finally, in the last case, we can see that, again due to the fact that the second case has a large vehicle capacity (2500 dm^3), only one vehicle can be used for 5 stores. In this case we can see that we have the biggest improvement from the initial solution, since customers with more than one order will be served only once, thus reducing the total time.

With the 20 percent of stores we can see that the previous trend is very similar, that is, in the same properties we have the best and worst results. There is in general an improvement of all the previous cases (10 % of stores) with a ratio of almost 2:1 due to the fact that having more stores there are fewer daily orders per shops. Another aspect is that there is a higher average improvement compared to the situation with 10 percent of shops.

Category: 100

In the case of 100 nodes with 10 percent of stores it is possible to observe from Figure 7.7 one difference with 50 nodes for highest values: the case 0 repeated orders, 2500 dm^3 capacity, and 30 min^2 map size (5) has not 1 vehicle per 5 shops as with 50 nodes but 1 per 3.33 stores. This signifies that 30 min^2 map size depends on the location of customers and shops and in the previous case with 50 nodes probably it was a case, in other words with these properties the ratio is between 3.33 and 5. Instead for the lowest values it is the same as before.

With 20 percent of stores there is one significant difference, the 30 min^2 map size has always 6.67 as ratio of shops per vehicle and 5 or less for 45 min^2. Instead previously there was not a big difference between 30 and 45 min^2, with in most of the cases 5 as ratio of shops per vehicle. This could be possibly explained by the fact that having more shops it is easier to split the stores with the vehicles.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,15)	5	23.42	4.61
2. (10,0,625,30)	3.33	26.58	3.68
3. (10,0,625,45)	2.27	16.95	3.08
4. (10,0,2500,15)	5	27.26	25.94
5. (10,0,2500,30)	3.33	27.55	8.09
6. (10,0,2500,45)	2.38	18.12	3.66
7. (10,30,625,15)	3.33	22.66	4.95
8. (10,30,625,30)	3.12	25.66	4.3
9. (10,30,625,45)	2	13.31	3.6
10. (10,30,2500,15)	5	32.13	20.89
11. (10,30,2500,30)	3.12	26.53	7.44
12. (10,30,2500,45)	2	13.31	4.13
13. (20,0,625,15)	10	38.78	55.04
14. (20,0,625,30)	6.67	44.5	67.48
15. (20,0,625,45)	5	41.48	64.13
16. (20,0,2500,15)	10	39.7	48.04
17. (20,0,2500,30)	6.67	44.33	71.52
18. (20,0,2500,45)	4.76	40.19	65.04
19. (20,30,625,15)	10	41.11	66.55
20. (20,30,625,30)	6.67	45	69.67
21. (20,30,625,45)	4	37.26	65.55
22. (20,30,2500,15)	10	43.38	58.45
23. (20,30,2500,30)	6.67	45.34	69.65
24. (20,30,2500,45)	4	35.77	65.53

Figure 7.7: Results of 100 nodes

Category: 150

With 150 nodes we can see from the Figure 7.8 that 100 and 150 nodes have really similar results, which continue to support the scalability of the algorithm. There are always the best cases with lower maps size 20 percent of stores. The only thing to change is the execution time which the increase from 100 to 150 nodes.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,15)	5	21.99	12.88
2. (10,0,625,30)	3.75	25.1	12.63
3. (10,0,625,45)	2.27	17.16	12.98
4. (10,0,2500,15)	5	25.83	42.98
5. (10,0,2500,30)	3.75	26.65	19.44
6. (10,0,2500,45)	2.27	16.84	12.04
7. (10,30,625,15)	3.75	21.33	17.06
8. (10,30,625,30)	3.12	21.83	15.85
9. (10,30,625,45)	1.92	11.54	11.31
10. (10,30,2500,15)	5	27.05	41.84
11. (10,30,2500,30)	3.12	22.46	19.35
12. (10,30,2500,45)	1.88	10.81	12.59
13. (20,0,625,15)	10	37.27	91.72
14. (20,0,625,30)	7.5	44.97	79.74
15. (20,0,625,45)	5	41.25	70.52
16. (20,0,2500,15)	10	38.35	101.53
17. (20,0,2500,30)	6.52	43.76	80.83
18. (20,0,2500,45)	4.55	38.77	71.53
19. (20,30,625,15)	10	39.95	90.29
20. (20,30,625,30)	6.25	43.08	79.19
21. (20,30,625,45)	4.29	38.22	89.76
22. (20,30,2500,15)	10	40.33	108.04
23. (20,30,2500,30)	6	41.51	81.74
24. (20,30,2500,45)	3.85	35.23	85.11

Figure 7.8: Results of 150 nodes

Category: 200

In this final category with 200 nodes there is again a similarity with the previous blocks, with some small differences: there is the case with 10 percent of stores, 0 repeated orders, 2500 dm^3 vehicle capacity and 15 min^2 map size (4) where the KPI shops per vehicle is greater than 5 (usual value), 6.25, which shows that it works also with more nodes, 200. The other aspect is that the lowest values with 10 percent of shops have a lower percentage decrease in time travel, this can be to the fact that increasing the number of nodes it should increase also the combinations of time and capacity constraint in order to find a better result.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,15)	5	23.19	64.21
2. (10,0,625,30)	3.85	27.69	58.81
3. (10,0,625,45)	2.38	18.52	55.27
4. (10,0,2500,15)	6.25	29.31	103.98
5. (10,0,2500,30)	4	29.72	72.35
6. (10,0,2500,45)	2.44	19.52	59.02
7. (10,30,625,15)	4	21.98	59.31
8. (10,30,625,30)	2.86	21.23	52.3
9. (10,30,625,45)	1.75	6.29	39.01
10. (10,30,2500,15)	5	26.82	112.19
11. (10,30,2500,30)	2.86	21.46	66.42
12. (10,30,2500,45)	1.89	8.28	50.04
13. (20,0,625,15)	10	38.91	106.04
14. (20,0,625,30)	6.9	42.63	91.31
15. (20,0,625,45)	4.88	41.86	88.07
16. (20,0,2500,15)	10	39.81	123.03
17. (20,0,2500,30)	6.9	43.11	92.51
18. (20,0,2500,45)	4.65	40.44	89.89
19. (20,30,625,15)	10	38.99	107.87
20. (20,30,625,30)	5.71	40.01	90.47
21. (20,30,625,45)	4.17	38.33	111.78
22. (20,30,2500,15)	10	39.81	127.14
23. (20,30,2500,30)	5.71	40.24	92.73
24. (20,30,2500,45)	4	37.02	110.86

Figure 7.9: Results of 200 nodes

Comparison: 50-100-150-200

In this sections there are the values analyzed previously visualized on graphs. There are 3 types of graph: Percentage decrease of total time, shops per vehicle and execution time. From the first graph in Figure 7.10, where there are all the combinations for each category with the percentage decrease, we can see that 50, 100, 150 and 200 nodes have a similar trend. Obviously it is not the same for each category but they have comparable results. It is also evident that with 10 percent of shops the results change a lot depending to the properties of each instance. On the other hand with 20 percent of shops the trend is more linear and each change of property does not affect a lot the percentage decrease of the route time. 20 percent of shops cases have also higher results compared to 10 percent of shops.

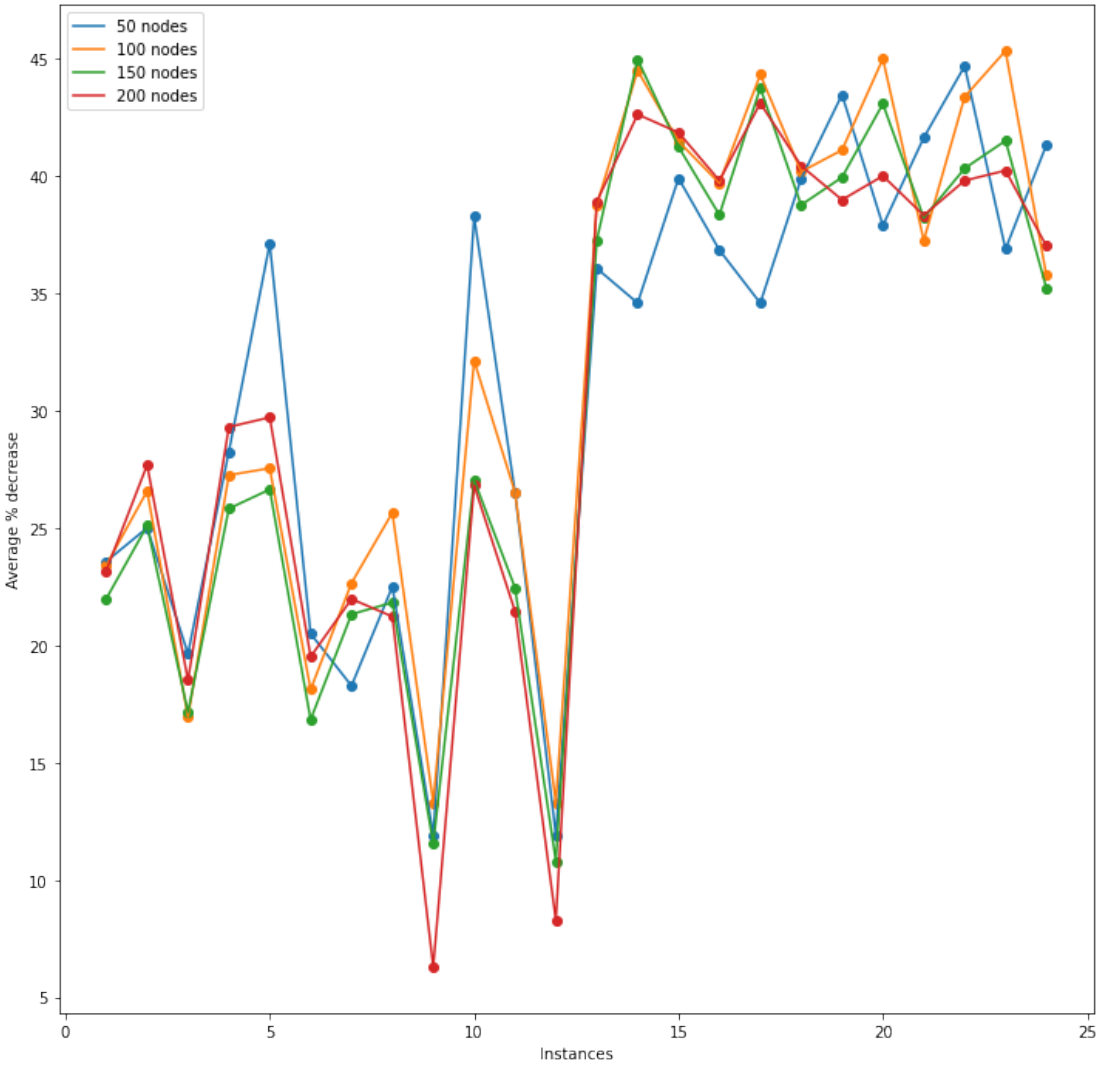


Figure 7.10: Percentage decrease: 50-100-150-200

In the graph in Figure 7.11 there are all the combinations for each category with the KPI shops per vehicle. Here the results of 50, 100, 150 and 200 nodes have a similar trend as for the percentage decrease. They have comparable results and also we can see that 200 nodes in some combinations have higher number of shops per vehicle, hence the algorithm works well also with high number of nodes with a reasonable amount of time. As before with 20 percent of shops the results are better than 10 percent of shops, but in this case with high variation also with 20 percent of shops. This is because the map size is a big constraint for the problem.

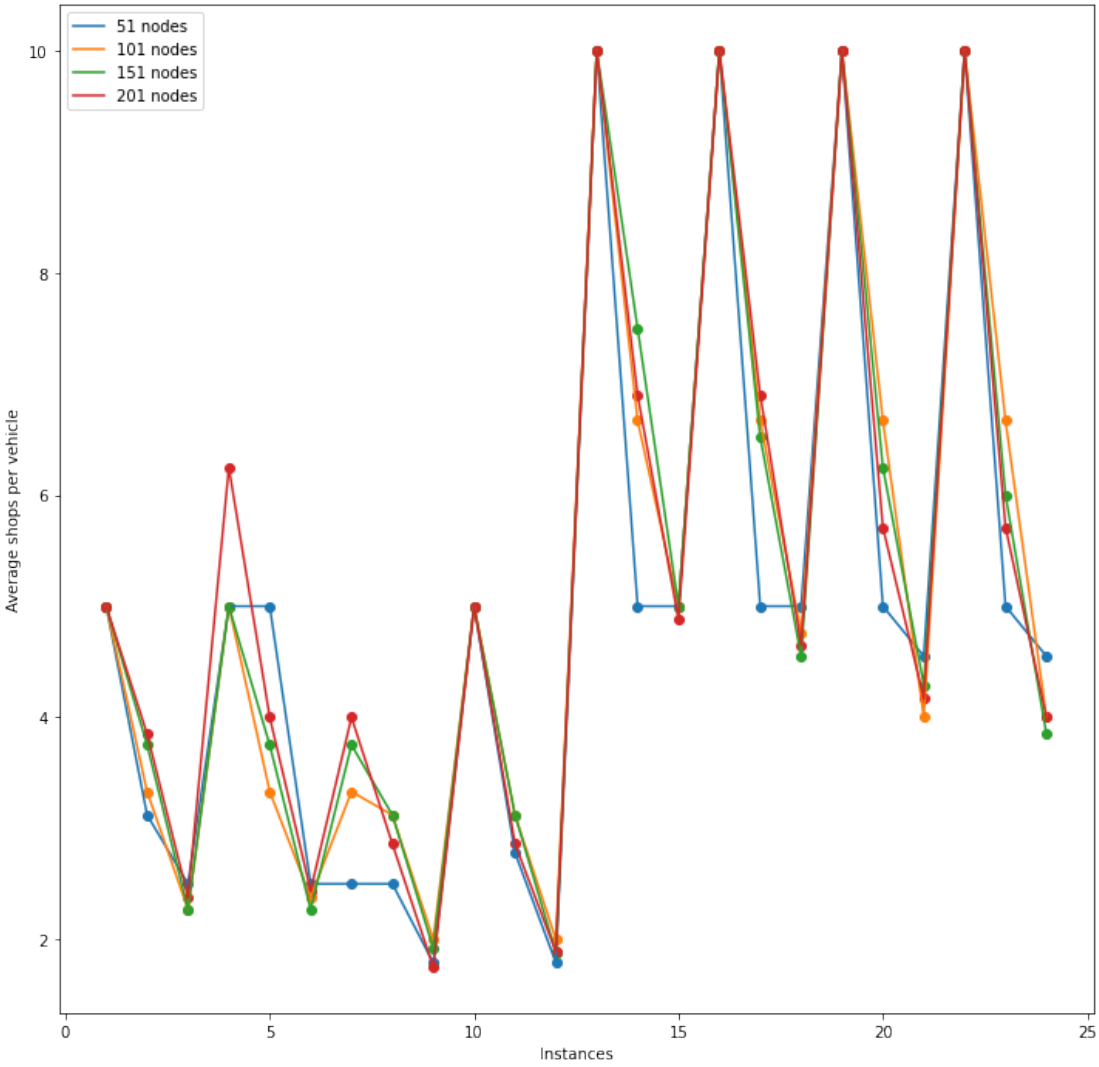


Figure 7.11: Shops per vehicle: 50-100-150-200

Finally, In Figure 7.12 it is possible to notice that increasing the number of nodes the execution time increase as well, and also that it increases when passing from 10 to 20 percent of stores for all the categories apart from 50 nodes which is constantly low. The execution times are always in the range of 120 seconds, which is one of the goals I was able to accomplish.

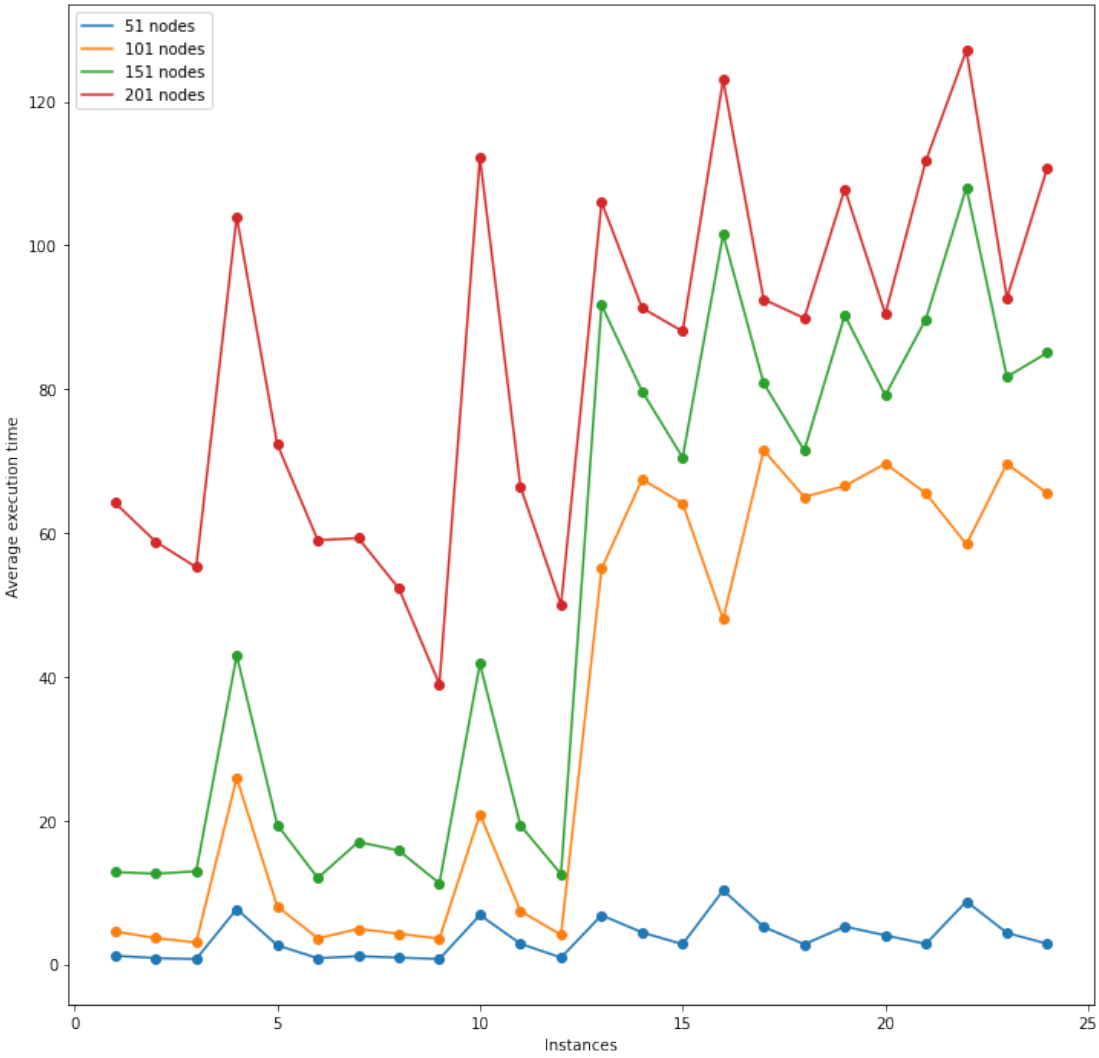


Figure 7.12: Execution time: 50-100-150-200

To sum up in Figure 7.13 there are the means of all the combination for 50, 100, 150 and 200 nodes. We can see that the results of the means are comparable between them with the only increase of the execution time related to the increased number of nodes. Also there is a new KPI, percentage decrease of travel time which does not keep in count of pick up and delivery time. It allows to show an higher improvement in the travel time which is the objective to minimize according to the number of vehicles. Since the algorithm works for all the nodes with a similar trend, probably it can be used also for a larger amount of nodes.

Number of nodes	Average shops per car	Average % decrease in total time	Average % decrease in travel time	Average execution time
50	4.41	32.12	34.75	3.89
100	4.63	32.77	35.46	36.01
150	4.59	31.19	33.82	53.28
200	4.61	31.22	33.87	86.39

Figure 7.13: Mean values: 50-100-150-200

7.3.2 Categories: 15-30-45

Categories: 15

In these sections there is the second classification considering more the map size instead of the number of nodes as before. I start with the map size of 15 min^2, which is the size closer to the problem of Links Foundation, since it should work for quarters of Turin. In this class there are the best results since many times it occurs that with the 10 percent of shops there are 5 shops per vehicle and 10 shops per vehicle with 20 percent of shops. In Figure 7.14 the lowest results are with 625 dm^3 vehicle capacity and 30 percent of repeated orders, hence the vehicle capacity becomes important when the customers have more orders from different stores and when the stores are the 10 percent of the total nodes.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,51)	5	23.58	1.23
2. (10,0,625,101)	5	23.42	4.61
3. (10,0,625,151)	5	21.99	12.88
4. (10,0,625,201)	5	23.19	64.21
5. (10,0,2500,51)	5	28.26	7.72
6. (10,0,2500,101)	5	27.26	25.94
7. (10,0,2500,151)	5	25.83	42.98
8. (10,0,2500,201)	6.25	29.31	103.98
9. (10,30,625,51)	2.5	18.28	1.17
10. (10,30,625,101)	3.33	22.66	4.95
11. (10,30,625,151)	3.75	21.33	17.06
12. (10,30,625,201)	4	21.98	59.31
13. (10,30,2500,51)	5	38.27	6.88
14. (10,30,2500,101)	5	32.13	20.89
15. (10,30,2500,151)	5	27.05	41.84
16. (10,30,2500,201)	5	26.82	112.19
17. (20,0,625,51)	10	36.07	6.86
18. (20,0,625,101)	10	38.78	55.04
19. (20,0,625,151)	10	37.27	91.72
20. (20,0,625,201)	10	38.91	106.04
21. (20,0,2500,51)	10	36.85	10.33
22. (20,0,2500,101)	10	39.7	48.04
23. (20,0,2500,151)	10	38.35	101.53
24. (20,0,2500,201)	10	39.81	123.03
25. (20,30,625,51)	10	43.47	5.28
26. (20,30,625,101)	10	41.11	66.55
27. (20,30,625,151)	10	39.95	90.29
28. (20,30,625,201)	10	38.99	107.87
29. (20,30,2500,51)	10	44.69	8.79
30. (20,30,2500,101)	10	43.38	58.45
31. (20,30,2500,151)	10	40.33	108.04
32. (20,30,2500,201)	10	39.81	127.14

Figure 7.14: Results of 15 min^2 map size

Category: 30

Increasing the map size, the KPI obviously decrease because the customers are spread in a wider map. The lowest results in Figure 7.15 are always with 625 dm^3

vehicle capacity and 30 percent of repeated orders, because the routes have a big restriction on the capacity. With the 20 percent of shops, sometimes it occurs that with lower capacity and the same other features there are better results. The reason why is that when many shops are assigned to one vehicle the algorithm can use only some combinations for the routes' order. For example if there are 8 stores assigned to one vehicle the permutations are 40320 and I take only 500 of them in order to have a solution with a reasonable execution time. With 15 min^2 map size it is different since there are many possible routes with 10 shops per vehicle that satisfy the constraints.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,51)	3.12	25.01	0.9
2. (10,0,625,101)	3.33	26.58	3.68
3. (10,0,625,151)	3.75	25.1	12.63
4. (10,0,625,201)	3.85	27.69	58.81
5. (10,0,2500,51)	5	37.12	2.68
6. (10,0,2500,101)	3.33	27.55	8.09
7. (10,0,2500,151)	3.75	26.65	19.44
8. (10,0,2500,201)	4	29.72	72.35
9. (10,30,625,51)	2.5	22.51	0.96
10. (10,30,625,101)	3.12	25.66	4.3
11. (10,30,625,151)	3.12	21.83	15.85
12. (10,30,625,201)	2.86	21.23	52.3
13. (10,30,2500,51)	2.78	26.53	2.9
14. (10,30,2500,101)	3.12	26.53	7.44
15. (10,30,2500,151)	3.12	22.46	19.35
16. (10,30,2500,201)	2.86	21.46	66.42
17. (20,0,625,51)	5	34.59	4.47
18. (20,0,625,101)	6.67	44.5	67.48
19. (20,0,625,151)	7.5	44.97	79.74
20. (20,0,625,201)	6.9	42.63	91.31
21. (20,0,2500,51)	5	34.59	5.25
22. (20,0,2500,101)	6.67	44.33	71.52
23. (20,0,2500,151)	6.52	43.76	80.83
24. (20,0,2500,201)	6.9	43.11	92.51
25. (20,30,625,51)	5	37.9	4.07
26. (20,30,625,101)	6.67	45	69.67
27. (20,30,625,151)	6.25	43.08	79.19
28. (20,30,625,201)	5.71	40.01	90.47
29. (20,30,2500,51)	5	36.89	4.4
30. (20,30,2500,101)	6.67	45.34	69.65
31. (20,30,2500,151)	6	41.51	81.74
32. (20,30,2500,201)	5.71	40.24	92.73

Figure 7.15: Results of 30 min^2 map size

Category: 45

In the category of 45 min^2 there are the worst results since the map size is the largest. The best and worst results are always with the same features. In Figure 7.16 with the 20 percent of shops it occurs more often that the vehicle capacity is not

a significant constraint, because with map size of 45 min^2 the time constraint is more affecting than capacity constraint.

Instances	Average shops per vehicle	Average % decrease in total time	Average execution time
1. (10,0,625,51)	2.5	19.65	0.77
2. (10,0,625,101)	2.27	16.95	3.08
3. (10,0,625,151)	2.27	17.16	12.98
4. (10,0,625,201)	2.38	18.52	55.27
5. (10,0,2500,51)	2.5	20.5	0.88
6. (10,0,2500,101)	2.38	18.12	3.66
7. (10,0,2500,151)	2.27	16.84	12.04
8. (10,0,2500,201)	2.44	19.52	59.02
9. (10,30,625,51)	1.79	11.92	0.78
10. (10,30,625,101)	2	13.31	3.6
11. (10,30,625,151)	1.92	11.54	11.31
12. (10,30,625,201)	1.75	6.29	39.01
13. (10,30,2500,51)	1.79	11.92	0.97
14. (10,30,2500,101)	2	13.31	4.13
15. (10,30,2500,151)	1.88	10.81	12.59
16. (10,30,2500,201)	1.89	8.28	50.04
17. (20,0,625,51)	5	39.91	2.83
18. (20,0,625,101)	5	41.48	64.13
19. (20,0,625,151)	5	41.25	70.52
20. (20,0,625,201)	4.88	41.86	88.07
21. (20,0,2500,51)	5	39.89	2.82
22. (20,0,2500,101)	4.76	40.19	65.04
23. (20,0,2500,151)	4.55	38.77	71.53
24. (20,0,2500,201)	4.65	40.44	89.89
25. (20,30,625,51)	4.55	41.64	2.87
26. (20,30,625,101)	4	37.26	65.55
27. (20,30,625,151)	4.29	38.22	89.76
28. (20,30,625,201)	4.17	38.33	111.78
29. (20,30,2500,51)	4.55	41.33	2.89
30. (20,30,2500,101)	4	35.77	65.53
31. (20,30,2500,151)	3.85	35.23	85.11
32. (20,30,2500,201)	4	37.02	110.86

Figure 7.16: Results of 45 min^2 map size

Comparison: 15-30-45

Here there is the comparison of the maps size, presented on graphs as in the previous section. In the percentage decrease of route time, in Figure 7.17 it is clear that with 10 percent of shops the map size of 45 min^2 is significant lower than the other two maps, however with 20 percent of shops the KPI is really close to all the maps.

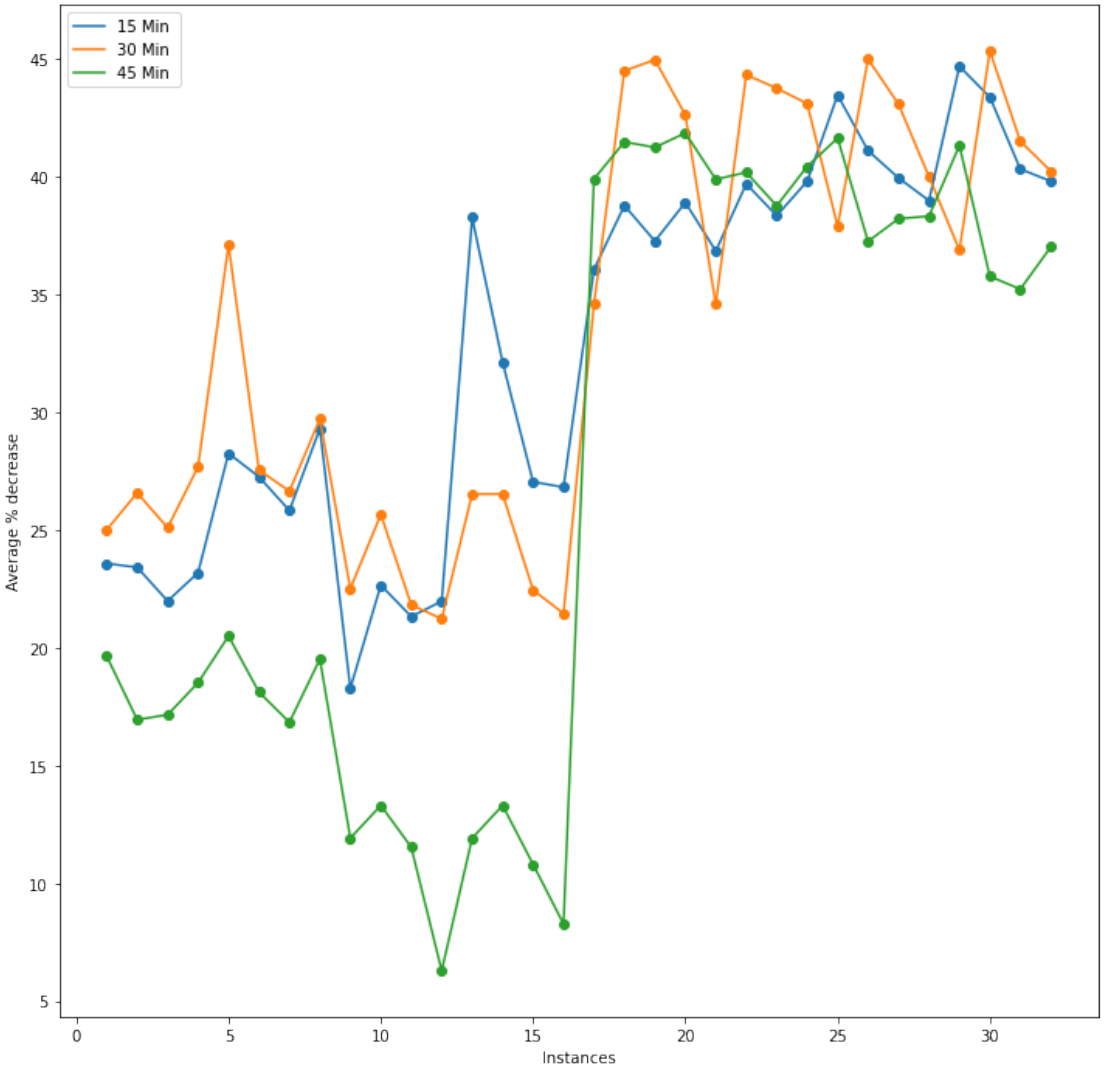


Figure 7.17: Percentage decrease: 15-30-45

In the shops per vehicle in Figure 7.18 it is more linear, for both 10 and 20 percent of shops there is a marked difference between the sizes of the maps. With 20 percent of shops the difference is more marked, since with 15 min^2 map size it is

always 10 shops per vehicle then from 5 to 7.5 shops per vehicle with 30 min^2 and finally from 3.85 to 5 shops per vehicle with 45 min^2. This is an important aspect which marks the result of 15 min^2 compared to the others in particular when the daily orders per shop are low (5).

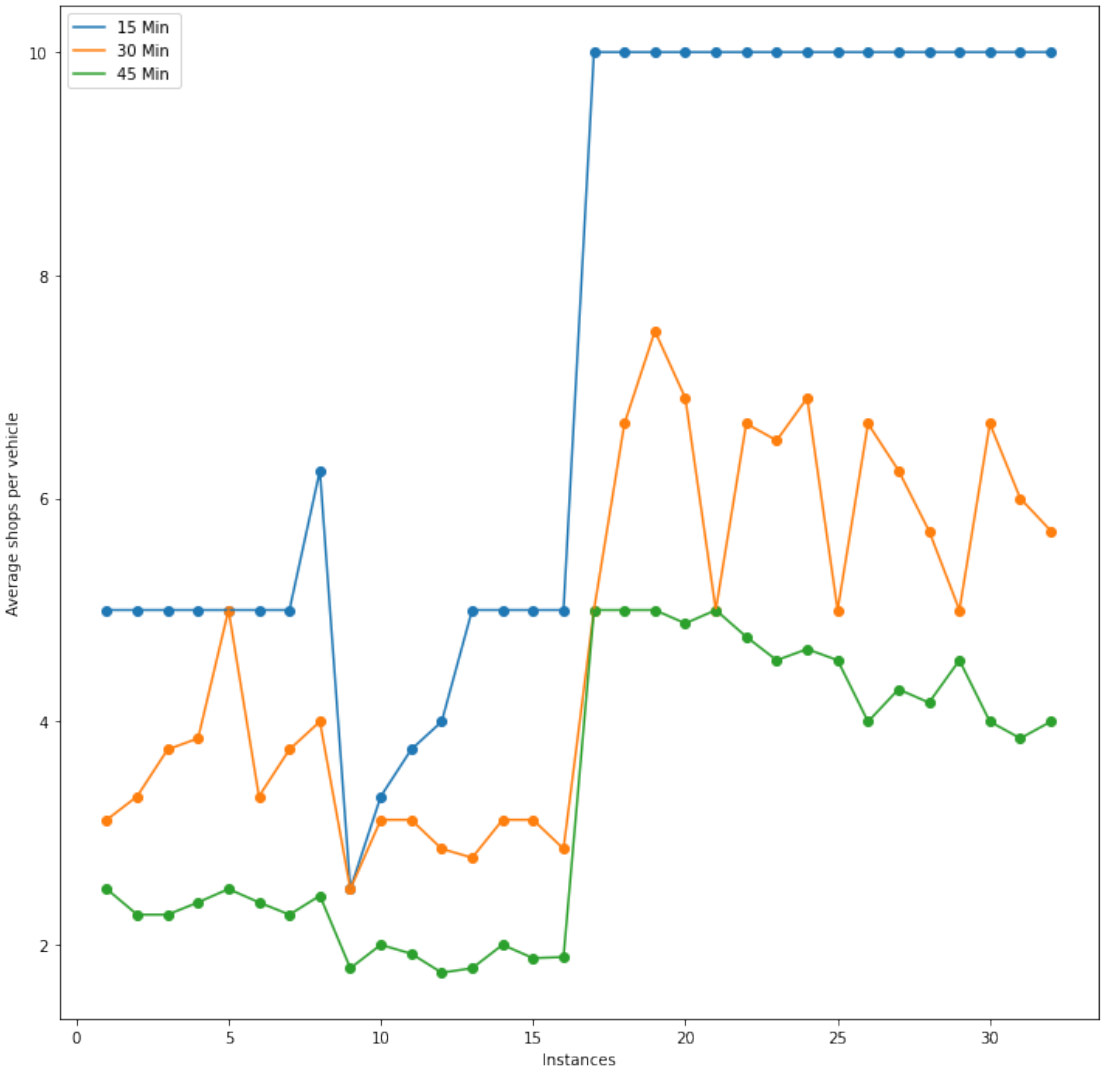


Figure 7.18: Shops per vehicle: 15-30-45

In Figure 7.19 it is shown the Execution time of the algorithm and there are the peaks with highest amount of nodes and with 15 min^2 because the algorithm takes more time when many stores are assigned to one vehicle. This is because with more shops per vehicle there are more permutations of the sequence of shops to pass by.

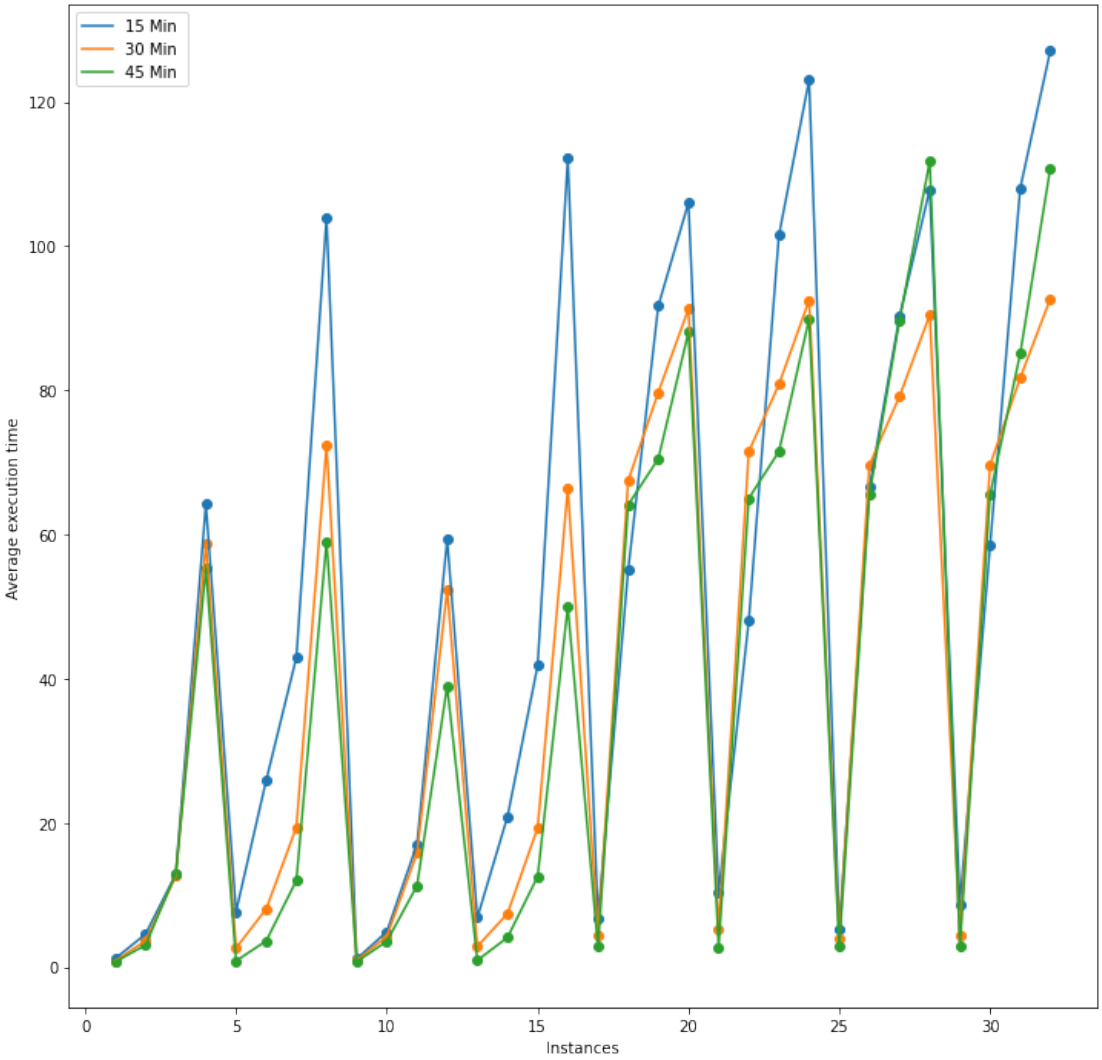


Figure 7.19: Execution time: 15-30-45

To conclude the comparison of 15 min^2, 30 min^2 and 45 min^2 in Figure 7.20 is is clear the negative trend between map size and shops per vehicle. Furthermore, the mean of the percentage decrease in total time is similar for 15 min^2 and 30 min^2, instead it is lower for 45 min^2 map size.

Dimension map	Average shops per car	Average % decrease in total time	Average % decrease in travel time	Average execution time
15	7.27	34.18	39.04	53.01
30	4.76	34.84	37.77	42.24
45	3.24	28.65	30.35	39.42

Figure 7.20: Mean values: 15-30-45

7.3.3 Comparison with higher execution time

I ran the algorithm for 16 instances with 200 nodes using more combinations of time and capacity constraint. I created for each vehicle more routes and also I did the Improved route for more solutions. It was to see the improvements of the results in comparison with the increase of execution time. In Figure 7.21 few times the vehicles used decreased, in particular with the feature 625 dm^3 vehicle capacity. The execution time increased a lot from 3 to 4 times the original value. Hence in these examples there is not an improvement in each instance, but with with some of them the improvement is relevant, considering that the execution time is still low.

Minutes, % stores, %Rep orders, Capacity	Average shops per vehicle final alg	Average shops per vehicle alg final with more time	Average % decrease in total time final alg	Average % decrease in total time final alg with more time	Average execution time final alg	Average execution time final alg more time
15,10,0,625	5	6.67	22.09	25.71	69.68	196.26
15,10,0,2500	6.67	6.67	30.34	30.34	104.23	383.98
15,10,30,625	4	5	22.71	25.6	58.06	159.4
15,10,30,2500	5	5	27.08	27.08	108.79	365.19
15,20,0,625	10	10	39.48	40.27	105.23	336.75
15,20,0,2500	10	10	41.06	40.95	130.48	453.61
15,20,30,625	10	10	38.64	39.34	107.47	353.48
15,20,30,2500	10	10	38.84	39.34	126.69	461.89
45,10,0,625	2.5	2.5	20.33	20.33	56.63	156.96
45,10,0,2500	2.5	2.5	20.39	20.39	60.47	160.36
45,10,30,625	2	2	10.73	10.73	36.12	87.83
45,10,30,2500	2	2	10.73	10.73	43.42	132.05
45,20,0,625	4.44	5	42.33	43	78.29	273.2
45,20,0,2500	4.44	4.44	41.93	41.69	79.85	248.79
45,20,30,625	4	4	38.49	38.65	83.9	289.78
45,20,30,2500	3.33	3.64	31.21	35.79	83.94	288.05

Figure 7.21: Increased combinations with 200 nodes

Chapter 8

Conclusions

The experiments reported have been performed on a 1,4 GHz Quad-Core Intel Core i5 CPU with RAM 8 GB. The algorithm is coded in Python 3.8 and the mathematical model of the algorithm is solved by GUROBI solver. The maximum execution time for all the tests is in the worst case around 125 seconds.

In this chapter I analyze all the instances differently from the previous chapter where they were analyzed only the extreme cases of the instances. In Table 8.1 all instances classes used to test the algorithm for each node size (50, 100, 150, 200 nodes) are presented. The first two columns represent the label of each instance analyzed, in the first one all instances having 10% of shops and the second one with 20% of shops. These labels, i.e. instance's names, help to read the next graphs: label 1 is the instance with 10% of shops, 15 min^2 map size, 0% of repeated orders and vehicle capacity 625 dm^3. For each instance there are 5 random simulations in order to get the average values of the algorithm's results.

In this section there is a new KPI: the percentage decrease of vehicles used. This substitutes the KPI, number of shops per vehicle, in order to have results more comparable with the KPI, the percentage decrease of route time. Starting from Figure 8.1 where it is shown a graph of the new KPI, the average percentage decrease of vehicles used for all instances described in Table 8.1. There are 4 distinct lines for 50, 100, 150 and 200 nodes, as in Chapter 7 with the classification according to the number of nodes. The x axis represents the Instances from 1 to 54 and y axis represents the percentage decrease of vehicles used. The percentage decrease of used vehicles has a similar trend for all the 4 lines, proving that the algorithm gives an output beyond the number of nodes. In this way this new deployment strategy can be used in cases with higher number of stores and customers, hence an area with higher population density, as a quarter of Manhattan.

In the left part of the plot there are the instances with 10% of shops, here results show that there is a much larger gap between the minimum and maximum decrease of used vehicles compared to the second half of the graph (20%). This shows that

10 % shops	20 % shops	Map size	% order repetition	Capacity
1	28	15	0	625
2	29	15	0	1250
3	30	15	0	2500
4	31	15	15	625
5	32	15	15	1250
6	33	15	15	2500
7	34	15	30	625
8	35	15	30	1250
9	36	15	30	2500
10	37	30	0	625
11	38	30	0	1250
12	39	30	0	2500
13	40	30	15	625
14	41	30	15	1250
15	42	30	15	2500
16	43	30	30	625
17	44	30	30	1250
18	45	30	30	2500
19	46	45	0	625
20	47	45	0	1250
21	48	45	0	2500
22	49	45	15	625
23	50	45	15	1250
24	51	45	15	2500
25	52	45	30	625
26	53	45	30	1250
27	54	45	30	2500

Table 8.1: Instances

with lower number of daily orders per shop there is lower variation in the percentage decrease of vehicles used. Another important element in the first half of the graph, is that increasing the map size, from 15 to 45 min^2, the decrease percentage drops, which is also present in the second half with a lower gradient. This relationship is important to understand that this problem is strictly affected by the dimension of the map in case the shops and customers are randomly distributed in the map when the ratio between customers and shops is about 10%. Again, in the left half of the graph it is possible to observe that the line with 50 nodes has some lower

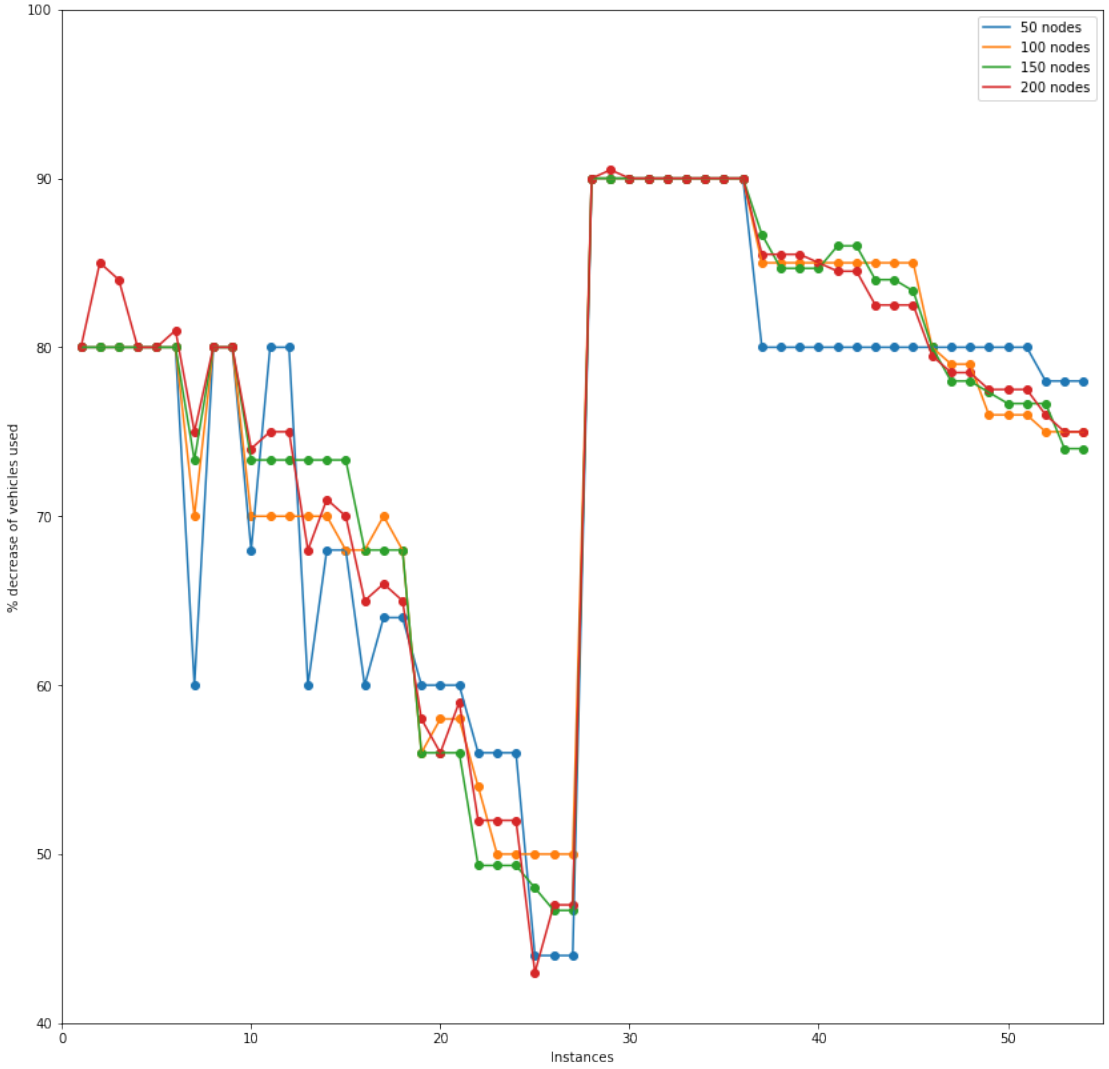


Figure 8.1: Percentage decrease of vehicles used

values with respect to the other lines for small vehicle instances (7,10,13,16), this is because with more shops it is not straightforward to have vehicles saved. In the second part of the graph instead there is no evidence that vehicles capacity strongly affects results since there are less customers assigned to a shop thus having more chance to decrease the load of the vehicle during the route passing to the customers. This last analysis is really important for the businesses because in certain scenarios having low vehicles capacity allows to save fixed costs for the vehicles and to have lower emissions.

In Figure 8.2 it is shown a graph of the average percentage decrease of route

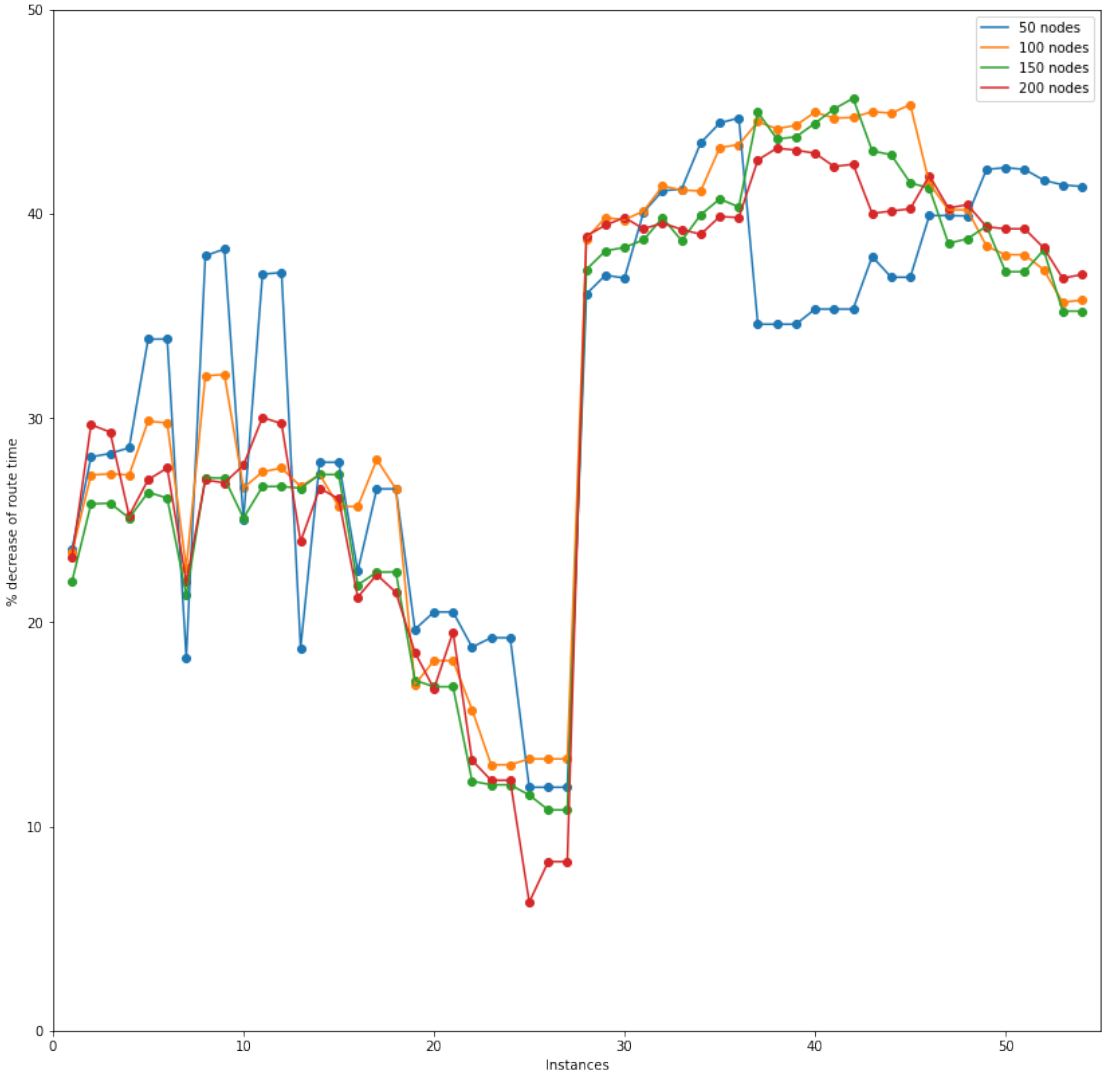


Figure 8.2: Percentage decrease of route time

time represented in the same way as the Figure 8.1. As in the previous graph it is notable the fact that in the first part of the graph (10 %) the percentage decrease of the global routes time has an higher interval of oscillation with respect to the right part (20 %). It continues to show that this solution has better results with low daily orders per shop. Apparently there is also the same trend with the maps size and the percentage decrease of the routes time, in particular with 45 min^2 map size the results are not clearly improving the initial solution. Instead, on the second part of the graph there is an higher linearity for all the instances which imply again that for instances with low daily orders per shop the algorithm finds a

considerable improved solution.

To sum up, instances with 20 % of shops (roughly 5 daily orders per shop without repeated orders) have the best improvements when the shared delivery service can be implemented in all map sizes. Instead when there are roughly 9 daily orders per shops (10 % of shops) it is preferable to have 15 min^2 map size or at most 30 min^2 map size in order to achieve relevant improvements for the businesses. Another relevant aspect is that the small vehicles have comparable results with the other vehicle sizes in 80% of the cases, except for 4 instances with 10 % of shop out of 18 instances (54 divided by the number of vehicle sizes). This somehow confirms that a local collaborative delivery service can be set up also with small vehicles.

In conclusion I have analysed a new variant of the VRP for *Pickup&Delivery* class related to the deployment of collaborative logistics models for local delivery service. After having developed the mathematical model, I designed and implemented an Heuristic Algorithm testing it on a large instances set. The results proved that for a local delivery service there is a relevant decrease in vehicles used and in routes time execution under specific conditions. This may be a great benefit for local shops allowing them to reduce logistics costs but also to serve customers that may not reach other-way. Moreover it is possible to reduce the traffic pollution and improve the environmental quality, contributing to the sustainability of the city.

Bibliography

[1] *What is O.R.?* URL: https://www.informs.org/Explore/What-is-O.R.-Analytics/What-is-O.R. (cit. on p. 1).

[2] Universita‘ di Roma “La Sapienza. *Appunti dalle lezioni di Ricerca Operativa*. 2004. URL: http://profs.sci.univr.it/~rrizzi/classes/RO/materiali/dispense/RO_displat.pdf (cit. on pp. 1, 3, 4, 6, 8, 23).

[3] Valeria Porcelli. *Pillole di ricerca operativa*. 2021. URL: http://www.ilfascinodellamatematica.com/home/pillole-di-ricerca-operativa/ (cit. on p. 2).

[4] Stephanie Glen. *Stochastic Model / Process: Definition and Examples*. 2021. URL: https://www.statisticshowto.com/stochastic-model/ (cit. on p. 3).

[5] Edmonds J. *Definition of Optimization Problems. In How to Think About Algorithms*. Cambridge: Cambridge University Press, 2008 (cit. on p. 6).

[6] Jeff Erickson. *Linear Programming*. 2018. URL: https://jeffe.cs.illinois.edu/teaching/algorithms/notes/H-lp.pdf (cit. on p. 11).

[7] Gruppo di Ricerca Operativa del Dipartimento di Informatica dell’Universita‘ di Pisa. *Appunti di Ricerca Operativa*. 2018. URL: http://groups.di.unipi.it/optimize/Courses/ROM/1819/Appunti1718.pdf (cit. on pp. 12, 13, 17).

[8] Marina Malatesta. «Metodi esatti i risoluzione per il problema di vehicle routing con finestre temporali». MA thesis. Padova: Università degli Studi di Padova, 2004 (cit. on pp. 15, 18–20).

[9] Deis Unibo. *Complessita*. URL: http://www.or.deis.unibo.it/algottm/files/complessita.pdf (cit. on p. 16).

[10] L. De Giovanni. *Methods and Models for Combinatorial Optimization*. URL: https://www.math.unipd.it/~luigi/courses/metmodoc1819/m02.meta.en.partial01.pdf (cit. on pp. 17, 21).

[11] Marco Liverani. *Elementi di Teoria dei Grafi*. 2014. URL: http://www.mat.uniroma3.it/users/liverani/doc/disp_oc_04.pdf (cit. on pp. 23, 27).

[12] Mary Attenborough. *Mathematics for Electrical Engineering and Computing,19 - Graph theory*. 2003. URL: https://www.sciencedirect.com/science/article/pii/B9780750658553500450 (cit. on p. 23).

[13] Adriano Corrente. «Techiche euristiche di clusterizzazione e percorso ottimo per risolvere un problema di raccolta e consegna di materiale». MA thesis. Padova: Università degli studi di Padova, 2014 (cit. on pp. 24, 29, 34, 35, 42, 44).

[14] Yuri Faenza. «Graphs and three algorithms for the TSP». In: (2018) (cit. on p. 25).

[15] JavaTpoint. *Graph Representations*. 2018. URL: https://www.javatpoint.com/graph-theory-graph-representations (cit. on p. 28).

[16] N. Bombieri F. Busato. *Chapter 7 - Graph algorithms on GPUs*. 2017. URL: https://www.sciencedirect.com/science/article/pii/B9780128037386000070 (cit. on p. 29).

[17] Michel Gendreau Christian Prins Thibaut Vidal Teodor Gabriel Crainic. «Heuristics for multi-attribute vehicle routing problems: A survey and synthesis». In: (2013) (cit. on p. 38).

[18] Sakaykumar J. *VEHICLE ROUTING PROBLEM AND ITS VARIANTS*. 2020. URL: https://www.linkedin.com/pulse/vehicle-routing-problem-its-variants-sajaykumar-j/ (cit. on pp. 40, 41).

[19] Wikipedia contributors. *Vehicle routing problem*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Vehicle_routing_problem&oldid=1001716538 (cit. on p. 40).

[20] Miguel Andres Figliozzi. *The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics*. 2011. URL: https://www.sciencedirect.com/science/article/pii/S1366554511001426 (cit. on p. 40).

[21] Ziaeifar Nickbeen Pichka Ashjari. «Open Vehicle Routing Problem Optimization under Realistic Assumptions». In: (2014) (cit. on p. 41).

[22] Aldà Michele. «Algoritmi Euristici per il Vehicle Routing Problem». MA thesis. Padova: Università degli studi di Padova, 2004 (cit. on p. 46).

[23] Elena Rocchi. «Progetto e sviluppo di un algoritmo per la pianificazione ottimizzata della distribuzione con viaggi sincronizzati». MA thesis. Bologna: Università di Bologna, 2012 (cit. on pp. 47, 48).

[24] Tiziana Delmastro. *Links Foundation*. URL: https://linksfoundation.com/en/about/commitment/ (cit. on p. 50).

[25] Gribkovskaia I. Laporte G. Berbeglia G. Cordeau J.-F. «Static pickup and delivery problems: a classification scheme and survey.» In: *TOP* (2007) (cit. on p. 53).

[26] Vidal T. Martins S. Ochi L.S. Souza M.J.F. Hartl R. Haddad M.N. Martinelli R. «Large neighborhood-based metaheuristic and branch-and-price for the pickup and delivery problem with split loads.» In: *European Journal of Operational Research* (2018) (cit. on p. 53).

[27] Speranza M.G. Archetti C. Nagy G. Wassan N.A. «The vehicle routing problem with divisible deliveries and pickups». In: *Transportation Science* (2015) (cit. on p. 53).

[28] Coelho L. C. Aziez I. Coté J-F. «Exact algorithms for the multi-pickup and delivery problem with time windows.» In: *European Journal of Operational Research* (2020) (cit. on p. 53).

[29] Roodbergen K.J. Jargalsaikhan B. Romeijnders W. «A Compact Arc-Based ILP Formulation for the Pickup and Delivery Problem with Divisible Pickups and Deliveries.» In: *Transportation Science* (2021) (cit. on p. 53).

[30] Iori M. Bruck B.P. «Non-elementary formulations for single vehicle routing problems with pickups and deliveries.» In: *Operations Research* (2017) (cit. on p. 53).