POLITECNICO DI TORINO

Master of science in Mathematical Engineering

Master's thesis

A top-down approach for the Dynamic Vehicle Routing Problem

Author:
Eleonora Vardè

Supervisor:
Paolo Brandimarte

Co-supervisor:
Giulio Zotteri

Academic year 2017-2018

Vorrei ringraziare tutti coloro che in un modo o nell'altro mi sono stati vicini durante questo percorso, e che mi hanno aiutata a superare i (numerosi) momenti di difficoltà. E vorrei dedicare questa tesi anche alla persona che ha forse creduto in me meno di chiunque altro, ma che alla fine, forse, si dovrà ricredere. Quella persona sono io.

Contents

Introduction

The use of software based on operation research and mathematical programming has become more and more popular over recent years, as part of the resource management.
The numerous real-world applications have shown that "The use of computerized procedures for the distribution process planning produces substantial savings (generally from 5% to 20%) in the global transportation costs." [22]
The impact on the overall economy is thus substantial, as the transportation process is a major component of the final product cost (up to the 20%).
The main reasons of the increasing success of these techniques are the introduction of more sophisticated models, and the technological development from both a software and a hardware point of view. While the former are able to describe more and more precisely the features of real-world problems, the second one has allowed a greater efficiency and speed in finding solutions to those models.
The aim of this thesis is to develop an algorithm for the solution of a dynamic vehicle routing problem, and it is motivated by a real-world application concerning the last-mile logistic of a big furniture distributor.
The *Vehicle Routing Problem* (VRP) involves finding the optimal routes that a fleet of vehicles has to perform in order to deliver the goods to a set customers, and it is one of the most important and thus studied optimization problems.
The thesis focuses on the *dynamic* VRP, where customers make their orders dynamically over time, and each day it is necessary to decide the pool of clients that are going to be served and the ones that should be left for a later day. Despite there are plenty of researches about the VRP, only few of them deal with the dynamic problem, which only in the recent years started being studied more deeply.
The approach we decided to take is denominated *Top-Down*, as it starts from

the whole daily queue of customers and then eliminates the least convenient ones, using a certain criterion. The remaining customers are those that will be served.
The first chapter describes the main features of the VRPs, its variants and a possible mathematical formulation. The second one focuses on the solution methods distinguished in the classical heuristics and the more recent meta-heuristics. The third one is dedicated to the description of the proposed algorithm and its three phases, compared to some benchmarks solutions. Chapter 4 describes the software used in the implementation and the main components of the program, while chapter 5 contains the analysis of the results obtained. Finally, in the conclusion, the limitations and the possible extensions of the proposed algorithm are outlined.

Chapter 1

Vehicle routing problem

The *Vehicle Routing Problem* (VRP) consists in finding the optimal set of routes that a fleet of vehicles has to perform in order to serve a given set of customers (see figure 1.1). Due to the numerous applications (and consequently its practical relevance) and its remarkable difficulty, it is one if the most important and studied combinatorial optimization problems. The problem was introduced by G. B. Dantzig and J. H. Ramser in 1959 [4]. They described a real-world application dealing with the delivery of gasoline from a bulk terminal to a large number of service stations, and they proposed the first mathematical programming formulation. Some years later, in 1964, Clarke and Wright [3] proposed a greedy heuristic improving the Dantzig-Ramser approach. Following those two articles hundreds of models and algorithms were developed in order to exactly or approximately solve different versions of the VRP. Thanks to the tecnological development, numerous software packages which implement these algorithms are now available.

1.1 Basics of the VRP

As introduced before, the aim of VRP is determining a set of routes on a given road network, each of them performed by a single vehicle starting and ending in its own depot, such that the customers' demands are fulfilled, the operational constraints are satisfied and the transportation cost is minimized. The road network is usually described through a graph $G = (N, A)$ in which nodes $N = \{0, ..., n\}$ correspond to the customers ($i = 1, ..., n$) and depot locations ($i = 0$, when there is only one depot), and arcs $A = \{(i, j) : i, j \in N\}$

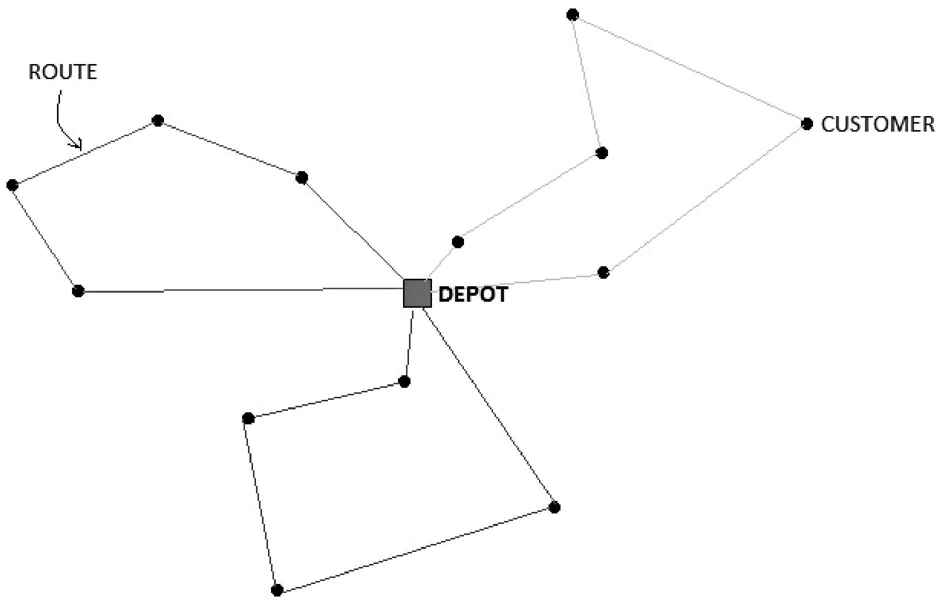


Figure 1.1: A picture illustrating the vehicle routing problem.

to roads. Each customer i orders a certain amount of goods (demand) usually denoted with q_i and often specifies a period of the day and/or a set of days (*time window*) in which he or she would like the goods to be delivered. Arcs can be directed or undirected depending on if the road can be traveled in only one or both directions. Each arc has a cost c_{ij}, that usually reflects its length, and it is associated with a travel time t_{ij}. Sometimes there is also a time associated to the delivery of goods (for example when a piece of furniture has to be assembled at the customer's house).
Routes are performed by a fleet of vehicles V. Each of them $(k = 1, ..., m)$ is based in one of the depots and has a fixed capacity C_k, which can be expressed in terms of volume, weight or number of pallets. In some cases a fixed cost is associated to the use of a vehicle.

1.2 Variants of the VRP

There are numerous different variants of the VRP, born in order to reflect the different features of real world problems. They mainly differ according to the type of constraints or objective functions that they contain.

1.2.1 Constraints

According to the nature of operational constraints, the vehicle routing problems are divided in different categories. The basic version of VRP is the *Capacitated* VRP (CVRP) in which each vehicle has a fixed capacity, known in advance. If the capacity constraint is substituted with a maximum route length/time constraint we talk about *Distance-Constrained* VRP (DVRP).
The VRP with Time Windows (VRPTW) is a variant of the CVRP in which each customer i specifies a time interval $[a_i, b_i]$ to be served in (*time window*).
Another extension of CVRP is the VRP with Back-hauls (VRPB), in which a certain amount of goods has to be delivered to the so-called $Line - haul$ customers, while other products have to picked-up from the $Back - haul$ customers. The VRP with *pick-up and delivery* (VRPPD) is similar to the previous one, but in this case each customer can both order some products and requires some others to be picked-up.
Figure 1.2 shows a graphical summary of the main VRP classes and their interconnections. This is only a simple and reductive description, as each of the above cited classes can be further divided and detailed, and many other variants are known.

1.2.2 Objectives

The objective of the VRP is determining the minimum *cost* routes. This *cost* can take into account of different aspects, the most common are: distance, time and customer satisfaction. Usually the most relevant purpose is finding routes of minimum length or such that the total traveling time is minimum. Other times, it can be important to minimize the customers' waiting time, the number of delays in the deliveries or the number of orders that are lost. Another example of cost could be the number of drivers' extra hours, as they are paid more than usual.
The objective function can take into account only one of this costs or many of them appropriately combined.

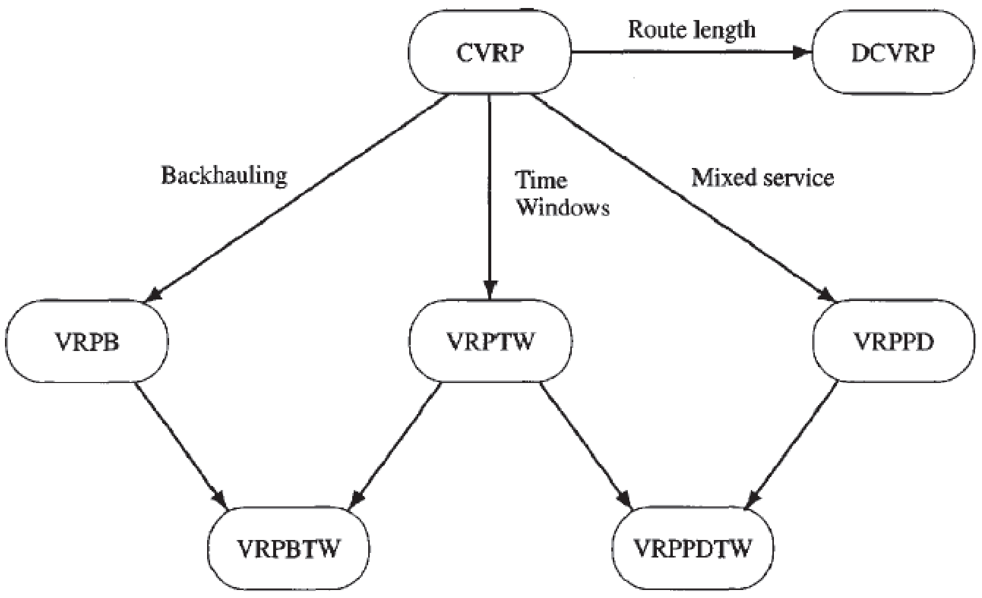


Figure 1.2: Classification of VRPs.

1.3 Mathematical formulation

Lots of different mathematical models describing the routing of vehicles have been proposed since the problem was first introduced. As an example, I will briefly describe a possible formulation of the VRP with time windows (VRPTW) as a multi-commodity network flow problem [15]. The main features of the VRPTW are a a set of customers C, a directed graph $G = (N, A)$ and a fleet of vehicles V. The graph consists of $|C| + 1$ nodes, where nodes $1, ..., n$ represents customers' locations, while node 0 represents the depot. The set of arcs A describes the connections between the depot and the customers and among the customers. A cost c_{ij} and a travel time $t_{ij} > 0$ is associated to each arc (i, j). All vehicles have the same capacity $Q > 0$ and a demand $d_i \geq 0$ is associated to each customer i. Customer i must be served in a time window $[a_i, b_i]$ $(a_i,\ b_i > 0)$, if the vehicle arrives before, it has to wait until a_i to deliver the goods. There are two sets of decision variables in the model: x_{ijk} and s_{ik}. $x_{ijk} = 1$ if vehicle k passes through the arc (i, j) and $x_{ijk} = 0$ otherwise. The decision variable s_{ik} denotes the time when vehicle k starts serving customer i, $s_{0k} = 0\ \forall k \in V$.
The objective is to find a minimum cost route for each vehicle, starting and ending at the depot, so that all customers are visited exactly once during

their service time window. The mathematical model is the following:

$$\min \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk} \tag{1.1}$$

subject to

$$\sum_{k\in V}\sum_{j\in N} x_{ijk} = 1 \quad \forall i \in C, \tag{1.2}$$

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ijk} \leq Q \quad \forall k \in V, \tag{1.3}$$

$$\sum_{j\in N} x_{0jk} = 1 \quad \forall k \in V, \tag{1.4}$$

$$\sum_{i\in N} x_{i0k} = 1 \quad \forall k \in V, \tag{1.5}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0 \quad \forall h \in C, \quad \forall k \in V, \tag{1.6}$$

$$x_{ijk}(s_{ik} + t_{ij} - s_{jk}) \leq 0 \quad \forall k \in V, \quad \forall i,j \in N, \tag{1.7}$$

$$a_i \leq s_{ik} \leq b_i \quad \forall k \in V, \quad \forall i \in N, \tag{1.8}$$

$$x_{ijk} \in \{0,1\} \quad \forall k \in V, \quad \forall i,j \in N. \tag{1.9}$$

Constraint (1.2) indicates that each customer is served exactly once and (1.3) is the vehicles capacity constraint. (1.4), (1.5) and (1.6) assure that all vehicles leave the depot, that after having reached a customer they leave for another one and finally they return to the depot. (1.7) describes the fact that a vehicle cannot visit a customer if it has not finished to deliver the goods to the previous one, it can be linearized as:

$$s_{ik} + t_{ij} - M_{i,j}(1 - x_{ijk}) \leq s_{jk} \quad \forall k \in V, \quad \forall i,j \in N. \tag{1.10}$$

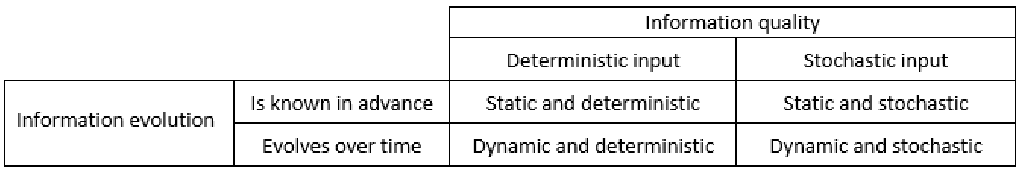
Here M_{ij} is a *large* positive constant. (1.8) and (1.9) are respectively the constraint on the service time and the integrality constraint.
In order to set an upper bound on the number of vehicles, a constraint of the following type can be added [5]:

$$\sum_{k\in V} d_i \sum_{j\in N} x_{0jk} \leq |V| \quad \forall k \in V, \forall J \in N. \tag{1.11}$$

Setting $a_i = 0$ and $b_i = M$ where M is a large scalar for all $i \in C$ the problem turns to a classical CVRP, while if $|V| = 1$ the model relaxes to a traveling salesman problem (TSP).

1.4 Extensions

The classical formulation of VRP doesn't take into account of two important aspects, which are relevant in real world application: evolution and uncertainty of information. For example, usually information is not known in advance: customers' requests arrive dynamically over time and for each of them a decision must be taken. In this case we talk about Dynamic VRP (DVRP). In other cases information is not deterministic but stochastic, for example we know that there will be a certain number of requests, in a certain area, at a certain time, with a certain probability. This is the case of Stochastic VRP (SVRP).
Obviously there are also versions that combines uncertainty and dynamicity. The table in fig.1.3 shows the taxonomy of VRPs by information evolution and quality [20].



		Information quality	
		Deterministic input	Stochastic input
Information evolution	Is known in advance	Static and deterministic	Static and stochastic
	Evolves over time	Dynamic and deterministic	Dynamic and stochastic

Figure 1.3: The taxonomy of VRPs by information evolution and quality.

Chapter 2

Solution Methods

The classical CVRP (and therefore its variants) belongs to the class of *NP-Hard* problems, which implies that there's no algorithm that is able to find a guaranteed optimal solution, in a polynomial time with respect to its size. For this reason, it's almost impossible to determine an exact solution for large-scale problems. Some algorithms can find optimal solutions to problems with about 50 or less customers; larger instances have been solved to optimality, but after a significant computing time. In practice, variations and additional constraints that must be taken into account in real-life contexts, make the vehicle routing problem even more difficult to be solved exactly.
These considerations explain why researchers focused on developing solution procedures based on heuristic algorithms. Heuristics are in fact designed to provide good feasible solutions within an acceptable computing time, but without a guarantee of global optimality. The families of heuristics for the VRP can be divided in two main classes: *classical* heuristics, developed mostly between 1960 and 1990, and *meta-heuristics*, the more recent ones [17]. The main difference among the two classes is that the first methods perform a limited exploration of the solution space, while the second ones deeply explore the regions of the search space, where a solution is more likely to be found. Since classical heuristics are able to produce good quality solutions within acceptable computing times and can be easily extended in the presence of additional constraints, they are still commonly used. On the contrary, meta-heuristics take significant computational times and can be hardly extended to different situations, but they are able to provide much higher quality solutions.

2.1 Classical VRP heuristics

Classical VRP heuristics can be classified into three categories: *Constructive heuristics*, *two-phase heuristics* and *improvement methods*.

2.1.1 Constructive heuristics

Constructive heuristics gradually build a feasible solution trying to keep the global cost as low as possible. They can be further divided in:

- *parallel* algorithms, in which routes are built together,
- *sequential* algorithms, in which routes are built one by one.

The two most used criteria to grow a route or merging routes are (see fig. 2.1)

- the *savings* criterion. It is based on the idea that if two customers i and j are served in two separate routes the total cost is $c_{i0} + c_{0i} + c_{j0} + c_{0j}$, while it is $c_{i0} + c_{ij} + c_{j0}$ if they are served by the same vehicle, with a saving $s_{ij} = c_{0i} + c_{j0} - c_{ij}$. The larger is this value, the more convenient is placing the two customer in the same route.
- the *extra-mileage* criterion. The *extra-mileage* is the incremental cost of inserting customer k between customer i and j, that is $e_{ikj} = c_{ik} + c_{kj} - c_{ij}$. The best insertion is the one which leads to the minimal extra-mileage.

The earliest and maybe most known example of constructive heuristic is the Clarke and Wright's algorithm [3], which uses the savings criterion in order to merge routes. Many variants and extensions [17] have been proposed since the creation of this algorithm, because it tends to form good routes at the beginning and worse ones towards the end, in particular circumferential routes.

2.1.2 Two-phase heuristics

In the two-phase heuristics the problem is decomposed in the two subproblems that characterize it: the *clustering* component and the *routing*

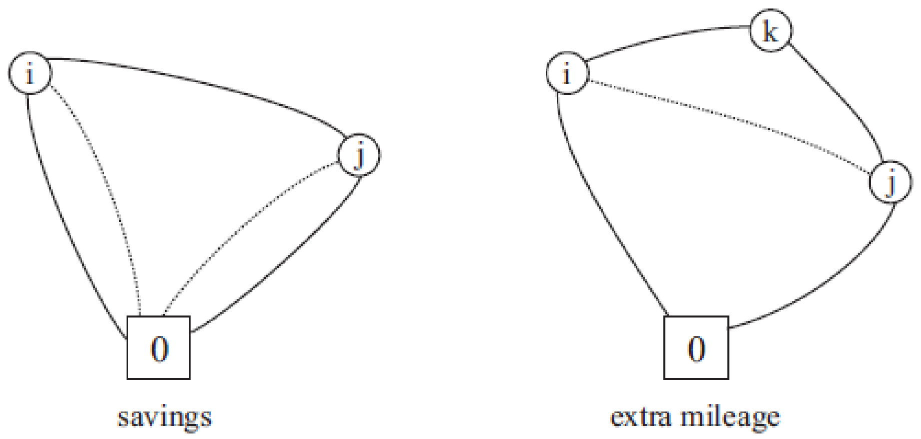


Figure 2.1: The two criteria to grow a route

component. The former deals with the division of customers into groups, each of which is performed by a different vehicle. The routing component instead, aims at finding the best route for each vehicle, that is the sequence in which customers have to be served and it implies solving a TSP problem for each vehicle. In according to the order in which these tasks are performed, two-phase heuristics are classified in:

- *Cluster-first, route-second*: customers are first divided into clusters and then a route for each vehicle is chosen.
- *Route-first, cluster-second*: first a tour including all customers is built, then it is divided into feasible vehicle routes.

The earliest implementations of the first approach are mainly based on geometrical considerations (one typical example is the *sweep* heuristic [10]), but these methods are satisfactory only if the distances between the points in the network are close to the Euclidean ones.
To overcome the above limitation, some more recent implementations rely on a partial mathematical formulation of the VRP, modeling the clustering component. I will outline the idea proposed by Fisher and Jaikumar in 1981 [8] that uses a combinatorial optimization model known as *generalized assignment*. The main features of this problem are: a set of n jobs and a set of m machines, and usually $m < n$ (less machines than jobs). Each machine $j = 1, ..., m$ is available for R_j units of time, and takes p_{ij} units of time and c_{ij} units of money to perform the job $i = 1, ..., n$. The objective is to find the

set of jobs to assign to each machine in order to minimize the overall cost. The mathematical formulation is the following:

$$\min \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij} y_{ij} \tag{2.1}$$

$$\text{s.t.} \sum_{j=1}^{m} y_{ij} = 1 \qquad i = 1, ..., n \tag{2.2}$$

$$\sum_{i=1}^{n} y_{ij} p_{ij} \leq R_j \qquad j = 1, ..., m \tag{2.3}$$

$$y_{ij} \in \{0, 1\}$$

The variable y_{ij} is 1 if job i is performed by machine j, 0 otherwise. The objective (2.1) is the overall cost, constraint (2.2) guarantees that each job is done by one machine only, while (2.3) that the machines time capacity is not exceeded.
This model can be used in the VRP setting interpreting the machines as the vehicles, and the jobs to do as the customers to serve. If the fleet of vehicles is homogeneous, that is all vehicles have the same capacity, $R_j = R$ for all $j = 1, ..., m$. We denote the customers' demand as d_i for all $i = 1, ..., n$. In principle we have the following formulation:

$$\min \quad \sum_{j=1}^{m} f(\mathbf{y_j}) \tag{2.4}$$

$$\text{s.t.} \quad \sum_{j=1}^{m} y_{ij} = 1, \qquad i = 1, ..., n, \tag{2.5}$$

$$\sum_{i=1}^{n} d_i y_{ij} \leq R, \qquad j = 1, ..., m, \tag{2.6}$$

$$y_{ij} \in \{0, 1\}.$$

Where $\mathbf{y_j}$ is the vector whose components are y_{ij}, $i = 1, ..., n$. In the objective (2.4), the function $f(\mathbf{y_j})$ represents the cost associated to the optimal tour of each vehicle j. As it is impossible to analytically write this function, it has to be approximated. A possible way could be:

$$f(\mathbf{y_j}) \approx \sum_{i=1}^{n} g_{ij} y_{ij}, \tag{2.7}$$

where g_{ij} describes the cost of assigning customer i to vehicle j. If we assume to have chosen m seeds σ_j (they have to be sufficiently distant from the deposit and one from the others), the cost coefficients can be approximated with the extra mileage of inserting customer i between the seed and the deposit, that is:

$$g_{ij} = c_{0i} + c_{i\sigma_j} - c_{0\sigma_j}. \tag{2.8}$$

Once this problem is solved, using Branch-and-bound for example, the TSP associated to each vehicle has to be solved in order to give a complete solution to the VRP problem.
Bramel and Simchi-Levi [1] proposed an approach based on a *concentrator location* formulation in order to avoid the manual choice of the seeds. We need the following variables:

$$x_j = \begin{cases} 1 & \text{if customer } j \text{ is selected as a seed,} \\ 0 & \text{otherwise;} \end{cases} \tag{2.9}$$

$$y_{ij} = \begin{cases} 1 & \text{if customer } i \text{ is in the route whose seed is j,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.10}$$

The model is the following:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{m}\sum_{i=1}^{n} g_{ij}y_{ij} + \sum_{j=1}^{n} v_j x_j && (2.11)\\
\text{s.t.} \quad & \sum_{j=1}^{n} y_{ij} = 1, \qquad i = 1, ..., n, && (2.12)\\
& \sum_{j=1}^{n} x_j \leq m, && (2.13)\\
& \sum_{i=1}^{n} d_i y_{ij}, \leq R, \qquad j = 1, ..., m, && (2.14)\\
& y_{ij} \leq x_j, \qquad i, j = 1, ..., n, && (2.15)\\
& y_{ij},\, x_j \in \{0, 1\}.
\end{aligned}$$

The coefficients $g_{ij} = c_{0i} + c_{ij} - c_{0j}$ are the extra mileages, while $v_j = 2c_{0j}$, which is the cost of the round-trip from the location of j to the deposit, is used as the cost of selecting j as a seed.

Constraint (2.13) guarantees that at most m seeds are chosen, (2.14) is the usual capacity constraint, while (2.15) relates to the fact that a customer which is not a seed cannot be assigned other customers. The objective is again to minimize the total cost of routing.

2.1.3 Improvement heuristics

Improvement heuristics aim at improving a current solution by a sequence of exchanges of nodes or edges. They can work on one route at time or on many route at the same time, in the first case improvement heuristics for the TSP can be used. Most of the TSP improvement procedures are related to the λ-*opt* technique developed by Lin in 1973, which consists of removing λ edges and rearranging the remaining segments [18]. If a better configuration is found, it is implemented, and the procedure goes on until a local optimum is reached or no more improvements are available.
The distinction between improvement methods and constructive methods is actually subtle, because often constructive heuristics include improvement steps at different stages.

2.2 Meta-heuristics

A *meta-heuristic* is a "master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality" (Glover and Laguna 1997 [13]). They are problem-independent techniques, that is they do not take advantage of any specificity of the problem. Usually they are not greedy and they may accept a temporary deterioration of the solution, which allows them to explore more deeply the solution space and thus to get a hopefully better solution. Meta-heuristics usually require a significant computational effort, but lead to solutions of better quality.
I will outline some examples of techniques that are commonly used in the VRP setting.

2.2.1 Tabu Search

The term *tabu search* was coined by Fred Glover in 1986 [12]. It is an intensive local search technique that uses memory of the information collected during the search, in order to effectively explore the solution space. Tabu search permits moves that deteriorate the current objective function value and that are chosen from a modified neighborhood of the current solution. The elements that are excluded from the original neighborhood are called *tabu*, and they are inserted into a *tabu list*, this procedure is useful to avoid cycling. In large scale problems, the tabu lists could save a significant amount of computing time, leading to improvements in search efficiency.

2.2.2 Ant Colony Optimization

Ant Colony Optimization was proposed by Marco Dorigo in 1992 [7], and it is based on the cooperation mechanisms that real ant colonies implement for finding short paths towards the food sources. In order to communicate they use *pheromone*, a chemical compound that attracts ants. If an ant finds a food source it will mark it, and the path to and from it, with pheromone. In addition, if an ant detects pheromone, it will follow it with a certain probability, which increases according to the number of ants that previously followed it. This leads to a growth in the number of ants that follow the same route, that becomes the favourite one, and that usually is the shortest or more convenient one.
This behaviour is imitated in the ant colony optimization algorithms, in which artificial ants moves inside the solution space marking the solutions with a weight, representing the amount of pheromone released.

2.2.3 Genetic Algorithms

Dating back to the early 1960s [9], the *Genetic Algorithms* (GA) are one of the most known examples of evolutionary algorithms, and they try to emulate the way in which the natural species evolve from one generation to another. Thanks to a certain scheme, the solutions are encoded as arrays of bits or character strings (*chromosomes*), which are then modified by *genetic operators*. The chromosomes are then selected according to the their fitness to be the solution to the given problem, in this way a new population is created. Each iteration, which leads to a new population, is called a *generation*. After some generations, the best chromosome provides the solution to the problem.

2.2.4 Simulated Annealing

Simulated Annealing (SA) is a probabilistic technique that draws inspiration from the metal annealing process. The annealing in metallurgy involves heating and controlled cooling of the material in order to increase the size of its crystals and reducing its imperfections.
At each iteration, the SA heuristic considers a state $s*$ in the neighborhood of the current state s, and decides whether to move there with a certain probability. This process leads the system to move to lower energy states and it is repeated until a certain *stability* is reached or until some constraints are violated.

Chapter 3

The proposed algorithm

In this chapter I will describe the real world application addressed in this thesis and the approach that will be followed in order to solve the problem faced. The first section focuses on the first topic, the second introduces the algorithm while the other three are dedicated to the different part of it.

3.1 Problem description

The practical application that I'm going to address in this report, is the last mile logistics of a big furniture distributor. *Last mile logistics* refers to the final step of the delivery process from a distribution center (the depot) to the end user (the customers). This topic is becoming more and more important because of the growth of online commerce. Last mile delivery costs are substantial, comprising 53% of shipping costs [6], and with the growing of free shipping customers are less willing to pay a delivery fee. In addition customers are becoming more demanding about the speed of the delivery. As a result, businesses are trying to develop new technologies and innovative supply chain models to ensure expedite deliveries, trying at the same time to cut their costs.
Last mile delivery can be a very tough task due to many factors: the geographical dispersion of customers' locations, the traffic and regulations in urban areas, the management of small orders and extremely big ones, the customers' requests,... The more constraints orders have to satisfy, the more computationally burdensome (sometimes unfeasible) becomes the vehicle routing problem that must be solved and the more expensive is the

delivery for companies.
In the real application that I'm going to consider, the delivery is *attended*, that is it must occur in the presence of the customer. For this reason the company usually proposes the customer a series of time slots for the appointment and he/she must chose the one he/she prefers. For the sake of the model simplicity I won't address this aspect and I will assume that customers cannot choose the time slot in which they would rather be served.
I will consider one depot, located near a big city. The surrounding land is devoid of natural barriers such as lakes or mountains, and it is provided with a good road network. Thanks to this feature, euclidean distances can be considered a good proxy of real distances.
Each client has a certain demand and can be served in a number of consecutive days that varies from one customer to the other. In the last day customers are considered *critical* or *urgent* and must be served otherwise the order is lost.
The fleet of vehicles is homogeneous, that is they all have the same capacity, and there is no fixed cost associated to the activation of a vehicle. For simplicity I will consider as capacity the maximum number of daily deliveries that a vehicle can carry out.
The most significant phase for the business is the choice of the daily customers pool, because choosing the right set leads to more efficient routes and thus more savings. For this reason it is important to elaborate an adequate strategy to accomplish this task.

3.2 The proposed algorithm

The approach that has been adopted consists of three phases:

- *Learning* phase: generation of a metric that is used in the executive phase to choose the pool of clients to serve.
- *Dispatching* phase: choice of the customers to be served each day.
- *Routing* phase: building of the optimal routes given the clients locations.

At the beginning of a certain day, only the customers who have made their order in the previous days are known, and no deterministic data about the future orders are available. For this reason it is important for the dispatching strategy to take into account of all the available information, as the choice of each day has impact on the following days.

3.3 Learning phase

The aim of this phase is to exploit the stochastic information about orders arrivals in order to generate a metric for the following phase. There are some ideas at the basis of the approach that will be used, and I will outline them using some examples. Let's suppose that a client c, who is located in an area with high density of customers, can be served today or in one or more following days. If today there are very few customers available near c, it is better to leave c for later. That's because it is likely that there will be soon some other requests in that area and they may be fulfilled together with c. Let's instead suppose that there are two clients c_1 and c_2: the former can be served today or in one of the four following days, the other one can be served today or tomorrow. It is reasonable to think that a good decision-maker would more likely serve $c2$ today rather than $c1$, as there are more opportunities of future aggregation for $c2$.
The learning phase produces some expected values for comparison, that take into account the customers' positions and their time left before the due-date (last day in which the order can be delivered), also called $lifespan$.
The region that is considered is a rectangular area of dimensions $205km \times 215km$. The information is aggregated in cells of dimension $5km \times 5km$. In addition we assume that the total number of customers in the area of interest follows a Poisson distribution of parameter λ, and they are assigned to a certain cell c with probability p_c. While λ is one on the parameters of the simulation, the probabilities p_c are given.
For each cell, and each value of lifespan, it is necessary to estimate the expected extra mileage of a customer to his nearest neighbour. In order to do that, we must simulate the behaviour of the system for a time period sufficiently long in order to have in each cell, and for each value of lifespan an adequate number of customers. As there are some cells in which the probability that a customer belong there is extremely low, this approach is not feasible as it would require a really time-consuming simulation.
For this reason, we exploit the *Poisson Arrivals See Time Averages* (*PASTA*) property. This property, valid for queuing systems with Poisson arrivals,

states that arriving customers see on average the system in the same condition as an external observer looking at it in an arbitrary moment of time.
Accordingly, what we decide to do, is to first build a customers portfolio containing the simulated queues for a certain period of time, and then computing the expected values of extra-mileage by generating a new customer in each cell. In particular, the first step is to build a fit-sample of the customers queues of each day in a certain time period. The function $portfolioGenerator$ (see section 4.2.2) carries out this task by generating and storing the customer queues at the beginning of the day and eliminating the customers that are served with a certain rule (by the *dispatcher*). For the sake of simplicity the chosen rule is to serve only the urgent customers. In addition, as it is needed for the customers queues to be independent in blocks, the customers queues are stored every second block of h_{max} consecutive days. The dimension of the block h_{max} matches the maximum number of a customer's *days-to-death*, that are the number of days remaining to the order expiration. The number of simulated days depends on the sample size that is needed, and on the transitory: in particular if the sample size is n and the transitory is T, the number of simulated days will be $T + 2n$. In this way the final number of daily queues that are stored is n.
After building the queues sample, the actual learning is performed. For each cell and each block of consecutive days, a customer is generated and his extra mileage (*em* for brevity) to the nearest neighbour in each of the queues of the days in the block is computed. As at this point no routes are built, the em is calculated with respect to the depot, that is we suppose that each customer i has its own route of the kind: $(0, i, 0)$ (where 0 represents the depot). It is recalled that the extra mileage em_{ikj} is the incremental cost when inserting customer k in the tour between customers i and j, in formulas:

$$em_{ikj} = c_{ik} + c_{kj} - c_{ij}$$

where c_{ab}, in this case, is the euclidean distance between a and b.
Afterwards, for each cell and block of days, for each value h from 1 to h_{max} the representative value of em is computed as the minimum value in the first h days of the block. This procedure leads to values of em that are decreasing with increased number of days-to-death (the less time remains, the more urgent is the order).
Finally the average extra mileages for each cell and possible value of days-to-death are computed by summing the values in the different blocks and

dividing the total by the number of blocks.

3.4 Dispatching phase

The objective of this phase is to define the pool of clients that have to be served day by day. In this phase no route is built, but only the set of customers is chosen. At the beginning of each day there is a queue of customers waiting for their order delivery. Among them, there are the critical or urgent ones: it is impossible to leave them for later, they must be served that day. Some simple logic that can be used to make the choice are the following two:

- serve ALL the customers until reaching the capacity,
- serve only the $URGENT$ ones.

Both of the logic above have their limitations and can lead to high operating costs. In particular, if we decide to serve all the customers, some of them could be very far from the depot and very isolated, probably we are losing future occasions of aggregating them with some other clients and the routes that we're building are not efficient. On the other hand it is equally not convenient to deliver only the urgent orders, as they are probably few and some not-urgent customers near the urgent ones could be served together with them.
What we would like to do with the proposed algorithm is to avoid the previous limitations, trying to choose each day a pool of customers which obviously contains the urgent ones but also the not-urgent ones that could be conveniently served that day together with the urgent ones.

3.4.1 Notations

In this section I would like to briefly introduce some notations that I will use in the rest of the report.
We denote as:

- U the set of urgent clients in the daily queue,
- NU the set of not-urgent clients in the daily queue,
- R the capacity of each vehicle,

- m the number of vehicles,
- $nCust$ the total number of customers in the queue,
- $nUrg$ the cardinality of the set U, while $nNotUrg$ the cardinality of the set NU,
- $totCap = m \times R$,
- em is a shorthand for extra mileage,
- when I say *take an appointment* with a customer in a certain day, it means that this customer is going to be served that day (we suppose that customers always accept to be served in the proposed day).

3.4.2 Algorithm description: the Dispatcher $Top\!-\!Down$

The algorithm we propose, uses a *top-down* approach to define the pool of clients to serve each day, in other words, first all the customers are divided into clusters, then the least convenient ones are excluded from the pool of clients that are going to be served that day. The way in which the algorithm works depends on the number of urgent customers:

- Case 1: $nUrg = totCap$. In this case only the urgent customers (all of them) are served and the not-urgent aren't even considered.
- Case 2: $nUrg > totCap$. In this case we have to decide who to serve (again only among the urgent ones), and to do this we solve the optimization problem presented in chapter 2 (2.11). The solution of this problem is the division of the customers into clusters. Afterwards, we compute the em of each customer to his nearest neighbour in his same cluster. If the customer is alone in his route, we suppose that his em to the nearest is a very high value (if he is alone in his route, it means that he's very far from the other customers, and he's maybe not convenient). Finally the customers are ordered according to increasing value of em to their nearest neighbour and an appointment is taken only with the first $totCap$ ones. The remaining orders are lost.
  It must be noted that this situation is very rare if the parameters of the simulation are finely tuned.

- Case 3: $nUrg < totCap$. In this situation there are two further sub-cases:

  - Case 3.1: $nCust \leq totCap$. First the optimization problem (2.11) is solved, second for each customer i we compute the em to his nearest neighbour in his same cluster (n_i) with respect to the depot (denoted with em_{in_i}).
  As in the learning phase, no route is built, so the values of em are always computed with respect to the depot (from now on, for the sake of simplicity I will denote with em_{ij} the value em_{i0j}).
  The value of em_{in_i} is compared to the average value referring to i's cell and days-to-death ($expEm_i$). We denote the difference $expEm_i - em_{in_i}$ as δem_{in_i}. The bigger is δem_{in_i}, the more convenient is to serve the customer, as it means that in that specific day his em_i is much lower than the average value. The values δem_{in_i} are ordered, and the appointment is taken only with the customers whose values are greater or equal than a certain threshold (which is a problem parameter).

  - Case 3.2: $nCust > totCap$. In this case we have two phases. In the first one we solve the optimization problem (2.11) and then, after computing the values of δem, we order the clients and select the top $totCap$. The second phase starts at this point, and it consists of solving another optimization problem with only the selected customers and again computing the values δem_{in_i} for each $i = 1, ..., nCust$. In order to avoid too long queues, it is possible to fix a value for the minimum number of customers we would like to be served in the "busy" days, as a percentage over the total capacity (p). The customers are again ordered according to decreasing δem, the top $p \times totCap$ are inserted in the pool while the remaining are selected only if their δem is greater or equal than a certain threshold, and until the capacity is still not saturated.
  The reason why the problem is solved twice and why we do a double selection, is because the first clusters that are formed, as the number of clients overcomes the total capacity, are not realistic (the number of vehicles has to be deliberately increased).

3.4.3 Dispatcher *DEM*

One of the starting points and benchmarks in developing our algorithm has been the master thesis of F. Patrone [19], a student of Politecnico di Torino who faced the same problem in a similar way in 2014. The main difference between his work and this one, is the dispatching phase, and I will now outline the main features of his dispatching strategy (the so-called *Dispatcher_DEM*).
The idea at the basis of his strategy is to consider the urgent customers as the seeds of the tours the are grown in parallel by adding the not urgent ones.
The algorithm works this way:

1. First an appointment is taken with all the urgent customers, which means that they are added to the set A, the *appointed*. The not-urgent ones are put is the set NA, containing the customers that still don't have an appointment.

2. It is determined the couple $a \in A$, $k \in NA$ such that δem_{ka} is maximum. Then it is identified $z \in NA$ such that $em_{kz} = \min_{j \in NA} em_{kj}$.

3. If $em_{kz} > em_{ka}$ or $\delta em_{kz} < thresh$, where $thresh$ is a parameter of the problem, it means that it is convenient to serve customer k together with a, so k is added to the set A. Go back to 2.

4. Otherwise, it is better to serve the two not appointed customers k and z together, and it is possible to form a group starting with them.
   For all the customers $n \in NA$, it is computed the minimum value of em with respect to all the members of the group (denoted with em_{ng}), then if the value $\delta em_n = expEm_n - em_{ng} > thresh$, then n is added to the group. The procedure goes on until the group dimension overcomes the capacity limit of the vehicle. The group is then added to A only if the total capacity constraint is not violated. Go back to 2.

This procedure has some limitations that we wanted to overcome by taking a different approach. The first and maybe the most significant one is the fact that there is some ambiguity related to the routes. In the dispatching phase, according to the author, no route should be built, but in *DispatcherDEM* some routes are actually built, even if they're called $pseudo - routes$ as

they will maybe not coincide with the final routes built by the router. Each pseudo-route is grown starting from an urgent customer considered as the seed, and the number of visited customers cannot overcome the capacity limit of a vehicle.
In addition, as there is not a phase in which routes can be merged, two or more urgent customers cannot be visited by the same vehicle in this phase. This is another great limitation of this algorithm as maybe the route that are built by the dispatcher may quite differ from the router's ones.

3.5 Routing phase

After having decided what is the group of clients we want to be served in a certain day, we have to build the *routes*, which means dividing the group into subsets and establishing the order in which the customers must be visited by each vehicle. The routing is performed by a VRP solver which exploits the method of *ejection chains*, a local search algorithm.
A fundamental aspect in all local search algorithms is the construction of a *neighborhood*, through the identification of all possible moves from one solution to another. In order to increase the performance of the algorithms, *compound* moves can being used. Compound moves are obtained by combinations of *basic* moves, that just consist of an insertion or exchange of vertices/arcs on the graph of the problem.
Ejection chain procedures were introduced by Glover in 1993 [11] for the TSP, and provide a technique to generate neighborhoods of compound moves. The fundamental feature of this technique is that each compound move is generated by complimentary steps, introducing certain elements and ejecting others. In other words, one step of the move creates disturbance (for example it violates a node degree by adding one or more edges), while the following one restores the balance.
In [21] we can find a possible application of ejection chains to the VRP setting, I will now briefly outline its main features.
The starting point is $S = (V, X)$, a partial graph associated to an initial solution. Rego, following Glover's perspective, defines an ejection chain as a sequence of triplets of consecutive nodes in a route. We denote a triplet as $(\underline{v}, v, \overline{v})$, where $v \in V$ is a vertex, and $\underline{v}$ and $\overline{v}$ are respectively its predecessor and successor in the route. An ejection chain of $l+1$ levels is defined over a

subgraph $L = (W, T)$, where T is a set of of $l + 1$ triplets denoted by:

$$T = \bigcup_{i=0}^{l} \{(\underline{v}^i, v^i, \overline{v}^i)\}.$$

An *ejection* consists of changing the position of one vertex and in an ejection chain each vertex v^i ejects the vertex v^{i+1}. The whole process transforms T into

$$T' = \bigcup_{i=1}^{l} \{(\underline{v}^i, v^{i-1}, \overline{v}^i)\},$$

and disconnects the graph S. The transformation is not a complete transition from a route set to another, as v^l doesn't belong to any route. It can be completed through the two following *connectivity* processes:

(a) *Type 1*: introduction of the set

$$T'_1 = (\underline{v}^0, v^l, \overline{v}^0);$$

(b) *Type 2*: replacement of the set $\overline{T}' = \{(v^l_r, v^l_s)\}$ with

$$T'_2 = \{(\underline{v}^0, v^0), ((v^l_r, v^l, v^l_s)\}.$$

The two processes are illustrated in picture 3.1, in the case of a three level ejection chain. The dotted lines represent T before the process of ejection, the back lines are the result of the complete transition.

On the basis of the chosen connectivity process, the resulting compound moves are named as:

(a) *multi-node exchange process*: represented by an ejection chain ended with type 1 connectivity process;

(b) *multi-node insert process*: ejection chain ended with type 2 connectivity process.

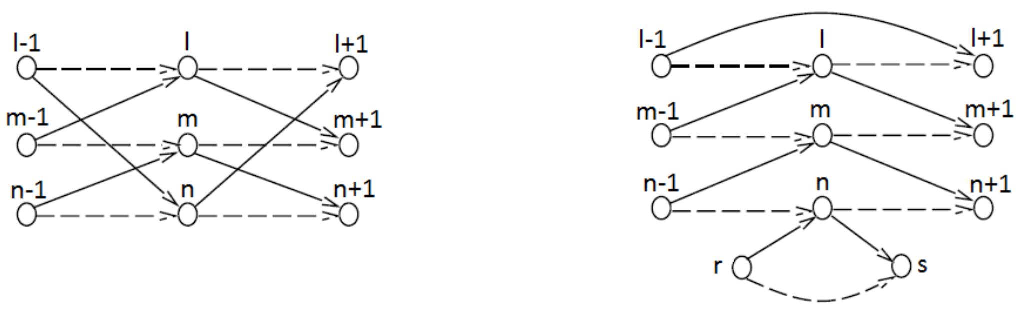


Figure 3.1: Connectivity processes.

Chapter 4

Implementation

This chapter focuses one the implementation of the program. All the programming languages and tools that are used in the different phases of the algorithm are specified in the first section. In the second and last section instead, I will outline the main components of the MATLAB program.

4.1 Used software

The project has been developed using MATLAB, but mostly with the use of the typical structures of Object Oriented programming. This approach in fact, allows to manage software complexity and it is particularly useful when developing and maintaining large applications and sophisticated data structures. One of the numerous advantages of MATLAB is the possibility of integrating the OOP concepts with the functionalities of procedural programming, which sometimes are more easily manageable.
The proposed dispatching algorithm, described in the previous chapter, includes the optimization model (2.11), which aims at dividing the customers into clusters. This problem is solved using Gurobi, a mathematical programming solver, which is faster and more powerful than the one embedded in MATLAB.
Finally the routing phase is performed by another solver, written in C++, which is part of VRPH, an open source library of heuristics for generating solutions to vehicle routing problems. This library was developed by Chris Groer and it's available on the website `https://projects.coin-or.org/VRPH`.

4.2 Main components of the program

4.2.1 Parameters and data

The file *grid.txt* contains the coordinates of each cell in the grid, and the probability that a customer belongs to that cell.
The class *Environment_parameters* is used as a container for all the parameters of the problem and of the different simulations.

4.2.2 Learning phase

The learning phase, described in chapter 3, is performed by the MATLAB script *learning* and makes use of the function $PortfolioGenerator$ that, as the name says, generate the portfolio of customers queues that ais used in the learning. The output of this phase are the maps of expected extra-mileages, stored in files named *EMdistributionXX.txt*, where XX indicates the parameter of the Poisson distribution used for generating the customers. The files contain a line for each cell and each line contains the expected em in that cell and for the different values of days-to-death.

4.2.3 Dispatchers

The class *Dispatcher* is the more general one and includes the main feature of a dispatcher, which is the algorithm that chooses the daily pool of clients. There are three different subclasses of the class *Dispatcher*: *DispatcherURG*, *DispatcherALL*, *DispatcherTopDown*. The first two ones are the benchmark dispatchers, used to compare the performance of our solution (the third one, described in section 3.4.2). The first chooses only the urgent customers in the queue, the second all the customers. In both cases, if the number of customers is greater than the total capacity, only some of them are selected according to the following parallel heuristic.
First m clients (as many as the number of vehicles) are chosen as *seeds*.
Denoting with:

- σ_k, $k = 1, ...m$ the seeds;
- c_{ij} the distance between nodes i and j;

σ_1 is chosen as the furthest node from the depot, while the others are those which have the greatest minimum distance from the other seeds. In formulas:

$$\sigma_j = \underset{i=1,...,n}{\arg\max}(\min\{c_{i0}, c_{i\sigma_1}, ..., c_{i\sigma_{j-1}}\}) \qquad j = 2, ..., m. \tag{4.1}$$

Once the seeds are defined, the route are expanded in parallel, selecting at each step the customer with minimum value of em for one of the open routes. If the insertion saturates the vehicle capacity, the route is closed and no other customer may be added to that route. When all the route are closed the procedure stops. However, in the case of *DispatcherALL*, there is the possibility that among the postponed orders, some one is urgent. If that happens, each remaining urgent customer is placed in the position of a not urgent customer that guarantees the minimum extra mileage.

4.2.4 Routing

The routing is performed by a solver named *vrp_ej.exe*, which is run inside the MATLAB function *VRPH*. In *VRPH*, first the input files containing the clients' coordinates and demands are defined, then the actual routing is performed (through *vrp_ej*). The function returns the cost (the total traveled distance) of the routing.

4.2.5 Daily simulation

The scripts *ProjectVRP_TopDown* and *ProjectVRP_Benchmark* implement the the dispatching and routing phases day by day. In both the projects there is a loop that simulates the time flow, each day a set of orders is generated and is added to the residual queue from the preceding days. Then the dispatcher (*topDown* in the first project and one of the two benchmarks in the second) chooses the set of orders that have to be satisfied and finally the router decides the vehicles routes. At the end of the time loop, all the statistics necessary to evaluate the algorithm performances are computed and stored in an output file named *Results_XXX*, where *XXX* represents the name of the dispatcher used (*TopDown*, *URG*, *ALL*). The considered performance metrics are: the average queue length, the total number of orders received and the number of delivered orders, the length of the residual queue at the end of the simulation, the average customers' waiting time, the average traveling cost per customer, and the total algorithm computing time.

Chapter 5

Analysis of the simulation results

This chapter is devoted the the experimental evaluation of the proposed algorithm on some simulated scenarios. The first section of the chapter focuses on the description of common assumptions parameters and data in the two scenarios, the other two sections describe the specificity of the two different scenarios and analyze the results obtained.
All the experiments were made with an Intel core i7-4510u 2 GHz machine.

5.1 Assumptions, parameters and data

In the two scenarios of simulation we're going to consider the region described in section 3.2, a rectangle of dimensions $205km \times 215km$ divided in cells of dimensions $5km \times 5km$, that form a grid containing 42 vertical cells and 44 horizontal cells. The cells coordinates, and the probabilities that a customer belongs to each of the cells, are contained in a file named *grid.txt*.
We're assuming that each customer can order only one item at a time and we're not considering its dimension, so the vehicles capacity is expressed only in terms of number of parcels they can carry, and not in terms of volume. It must be noted that this is not true in the case of a furniture distributor, because furniture are sometimes very voluminous.
In the real application, each day customers are called and they can decide whether accepting or rejecting to be visited. In the model we're assuming that they always accept, because we want to analyze the impact of the different politics regardless of the variability introduced by the customers decisions.

Each order can have a life-span that is a number of consecutive days uniformly distributed in the interval $[0, 5]$.
We assume that there are 20 vehicles, with the same capacity of 5 items per vehicle, the maximum number of customers that can be visited in a day is thus 100.
For all the simulations the time horizon is of 100 days, and we decided to truncate the simulation after the 100 days are finished. The residual queue of customers as the ones of the first 10 days, are not considered in the computation of the performance metrics. The simulated time period is adequately long for guaranteeing that this decision doesn't affect heavily the results.

5.1.1 Parameters of the router

As explained in the previous chapter, the chosen router is based on a local search heuristic, the *ejection chains* method. The following parameters have been set in the router.

- Initial solution: *Sweep.* The initial solution is obtained using the *sweep* method, a cluster-first, route-second heuristic (see section 2.1.2). This algorithm works this way: in the bi-dimensional map of customers and depot locations, the depot is considered as the center of rotation of a half-line passing through a chosen node in the map. The ray *sweeps* through the nodes surrounding the depot, rotating in counter-clockwise or clockwise direction, and assigning the encountered nodes to a cluster, until the vehicle capacity is not saturated (see image 5.1). After completing the clustering phase, a TSP is solved inside each cluster in order to route the customers.

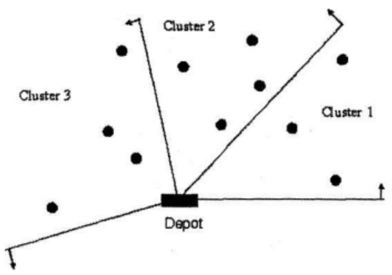


Figure 5.1: A picture illustrating the *sweep* heuristic.

- Number of ejected nodes: 5. This is the number of nodes ejected at each iteration of the ejection chain algorithm.

- Number of attempts: 700. This represents the number of attempts to improve the current solution. Increasing this value leads to better final solutions, but increases the computing time.

- Search type: *Regret search.* The *regret* represents the opportunity loss at each step, that is the profit gap between the current state and a possible future state. At each step it is chosen the direction that leads to the minimum regret.

5.2 Scenario 1

In this scenario we consider a depot located in the area with highest concentration of customers. In particular the depot coordinates are [122.75, 154.1]. Figure 5.2 shows an example of the customers' distribution in a certain day. In particular the black square represents the depot, the green dots are the customers chosen by the dispatcher, the red ones represent the urgent ones, and finally the black dots are the customers left for later.

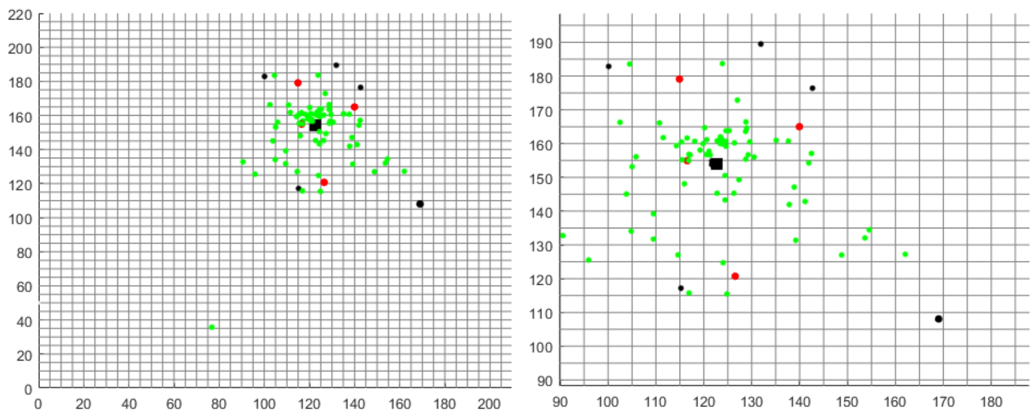


Figure 5.2: On the left the customers' locations map, on the right a zoom of the most crowded area.

5.2.1 Tuning of the threshold

The *DispatcherTopDown* uses a threshold to decide whether the benefit of visiting a customer in a certain day is sufficiently high with respect to the expected value in that area. The first simulations are needed to choose the optimal value for this threshold. In order to accomplish this task, we repeat the simulation with 3 different values of λ (the Poisson distribution of arrival parameter), and 8 different values of the threshold. Figure 5.3 shows how the average cost per customer (the average number of kilometers traveled to reach a customer) varies with the value of the threshold in the three different scenarios of demand. In addition, in order to make a comparison, the three graphs also show the prices obtained using the 'dummy' dispatchers. From the figures, we can observe that the minimum price is obtained by fixing the threshold at -8. However, in all the cases, the price obtained using $DispatcherTopDown$ is lower than the one obtained with the 'dummy' dispatchers.
Sometimes the number of kilometers traveled is not the most important factor in a business decision, as focusing only on this value can be at the expense of the customer satisfaction.
The additional advantage of the proposed dispatcher is that by varying the threshold we can also set the levels of two other important parameters that are the average customers' waiting time before the service and the average number of customers in the queue at the end of the day. The two graphs in fig. 5.4 show the behaviours of this two performance indicators with respect to the different values of the threshold.
We can see that with a higher threshold the average waiting time before the service and the average level of the residual queue using the $DispatcherTopDown$ get higher. In the case of the $DispatcherURG$ the two metrics tend to be very high as the customers are left waiting for the longest time possible. On the contrary, using the $DispatcherALL$, they tend to be nearly equal to zero as customers are served as soon as possible.

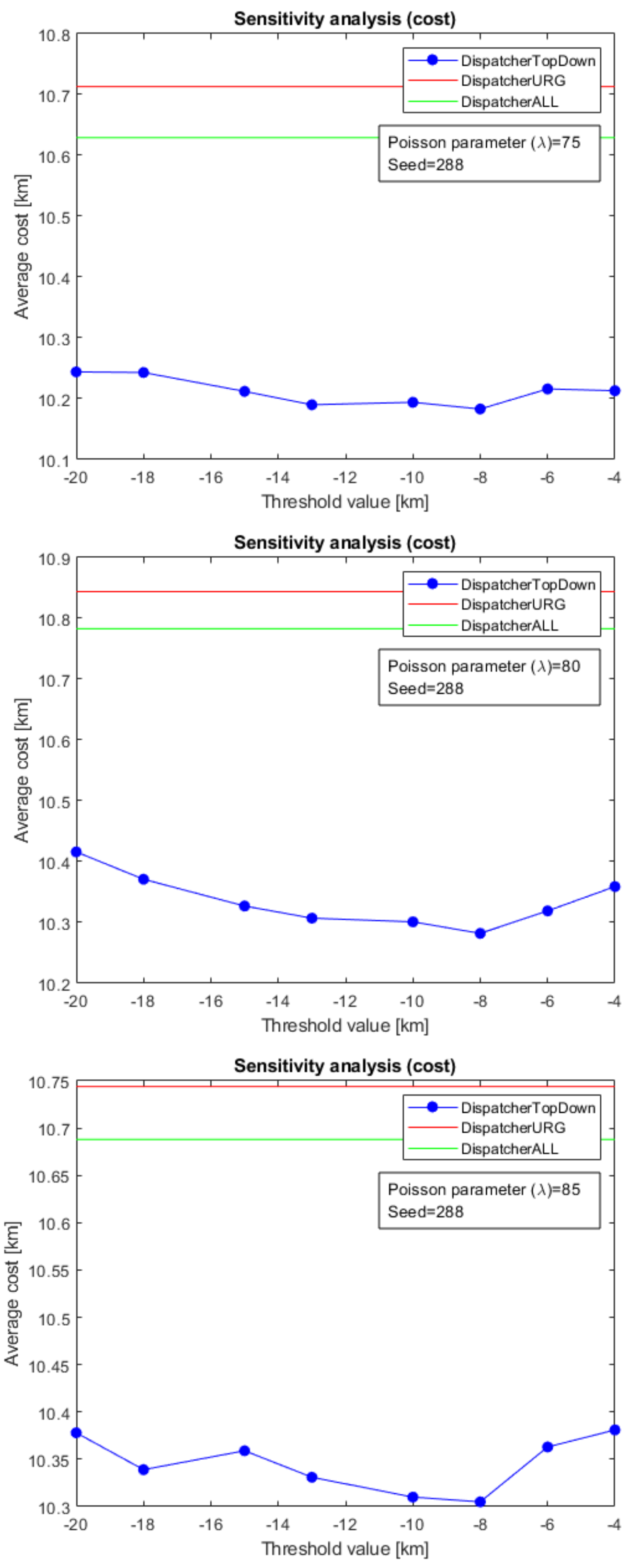


Figure 5.3: Sensitivity analysis for tuning the threshold

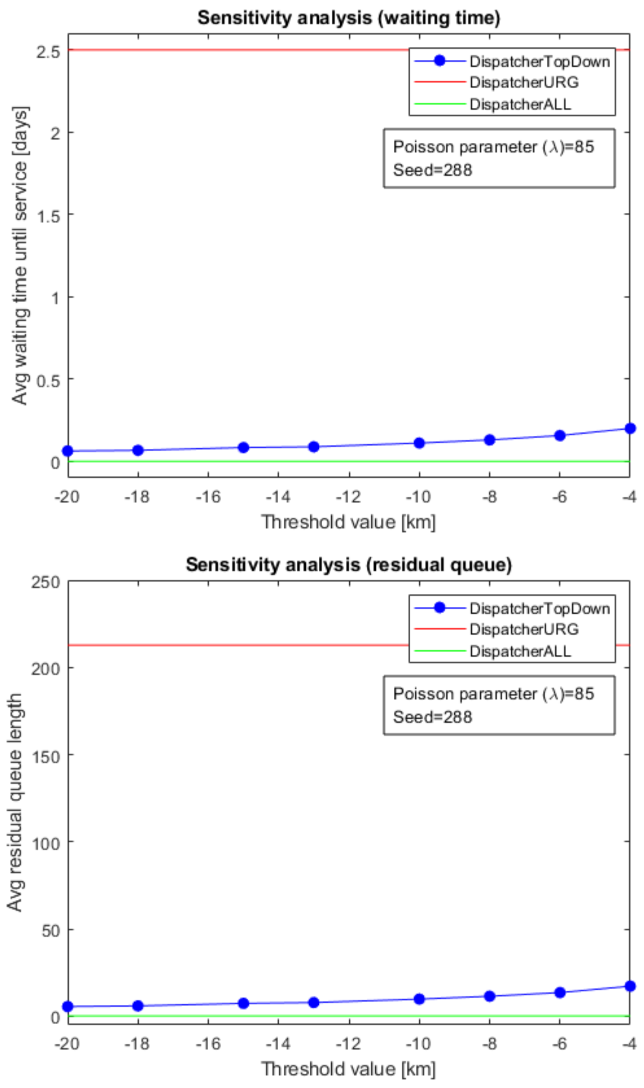


Figure 5.4: Waiting time and residual queue with varying threshold

5.2.2 Comparison with the benchmarks

In order to understand whether the *top − down* approach is efficient, we compare its performances with the ones obtained using the *dummy* strategies. The three graphs in figure 5.5 show the behaviour of the average cost per customer with different values of demand, using the different dispatchers. The threshold is set at -8, as in the previous section we saw that this value is the optimal one, and the simulation is repeated with three different seeds, as we need to understand if the efficiency is not due to particular conditions of a specific simulation.
In all the graphs we can observe that the cost tends to decrease with the increase of the average number of customers (λ). This phenomenon is due to the fact that a higher λ leads to an increase in the concentration of customers that get closer together, the distances in the routes are thus lower. In addition, we can see that the use of the *DispatcherTopDown* implies a saving of almost 0.5 km per customer on average, with respect to the dummy dispatchers. The advantage of using our algorithm gets lower with higher values of demands, that's because the benchmark strategies are more performing with a higher density of customers.
Using *DispatcherURG* or *DispatcherALL* results in almost the same cost. This is probably due to the absence of any logic in the choice of customers in both the strategies.

Figure 5.6 shows the percentage of saving using *DispatcherTopDown* instead of *DispatcherURG*, with the same three different seeds as before to generate the random sequence of customers. The savings are computed as:

$$\frac{cost(URG) - cost(TopDown)}{cost(URG)}$$

Here *cost*(*URG*) and *cost*(*TopDown*) denote the costs resulting from the use of *DispatcherURG* and *DispatcherTopDown* respectively. From the graph we can notice a percentage of saving that goes from a minimum of $\sim 2\%$ to a maximum over 6%.

In figure 5.7 we can instead observe the behaviours of the two other performance metrics (the average residual queue length and the average customer's waiting time in the queue) introduced in the previous section.
The waiting time and the length of the queue tend to increase in all the

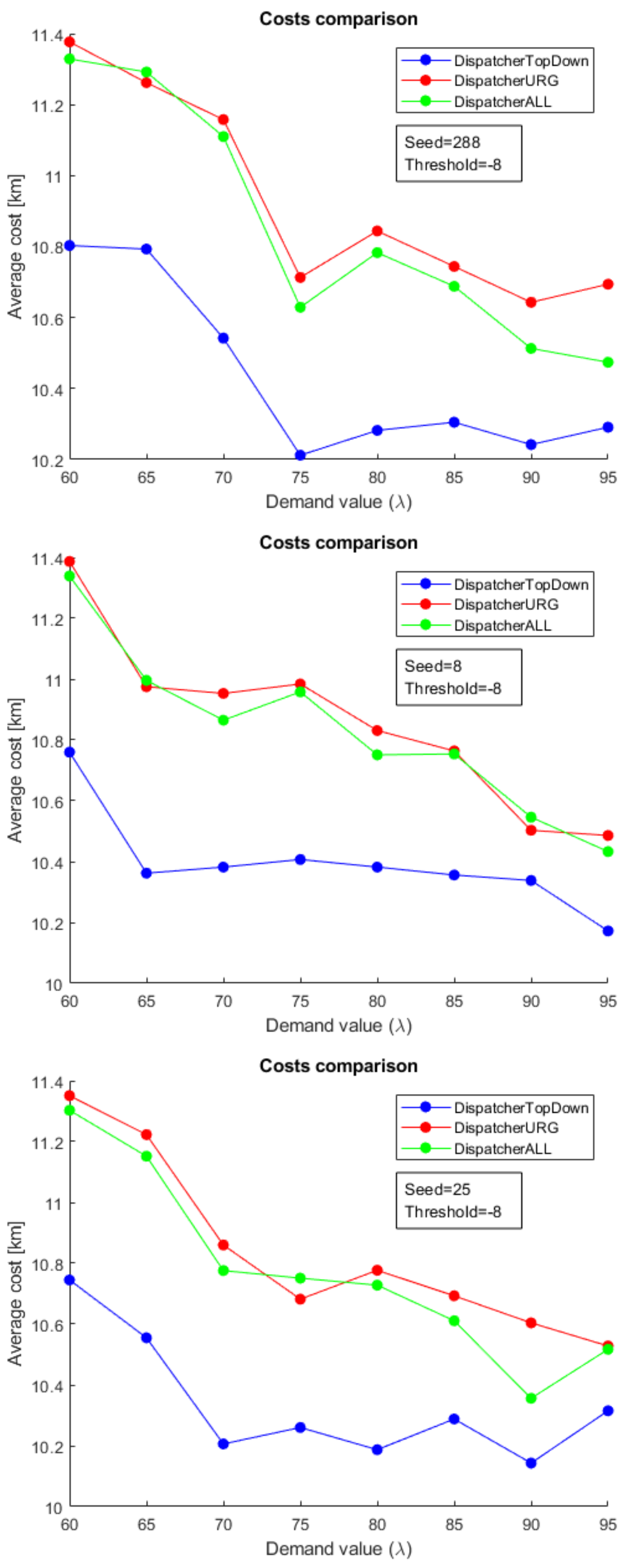


Figure 5.5: Average costs comparisons

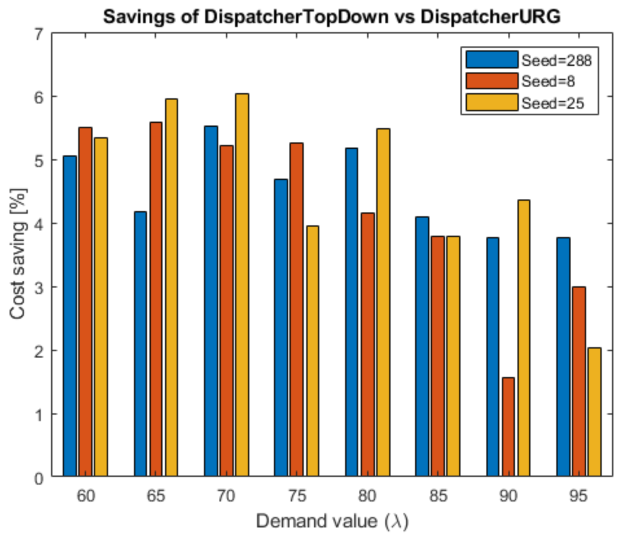


Figure 5.6: % Saving TopDown over URG

three cases (sometimes really slightly), when the number of customers is higher. Serving all the customers translates into almost zero waiting times and queues, while serving only the urgent ones leads to long queues and long waiting times and sometimes implies the loss of some orders (if the number of urgent order overcomes the total available capacity). The advantage of our strategy is that it combines a lower cost with very low waiting times for the customers and thus shorter queues.

We also want to compare the performances of *DispatcherTopDown* with those obtained using *DispatcherDEM*. In figure 5.8 we can see the results of the analysis. In all the simulations we used seed=8, cut-off threshold= -8 for the *DispatcherTopDown* and -4 for the *DispatcherDEM*. The three graphs show respectively the behaviour of the average number of km traveled to serve a customer, the average number of days each customer waits, the average length of the queue at the end of the day, with varying demand. The average cost in the case of *DispatcherTopDown* is a little higher than the in the other case, but the higher cost is justified by the presence of lower customers' waiting times and shorter residual queues.

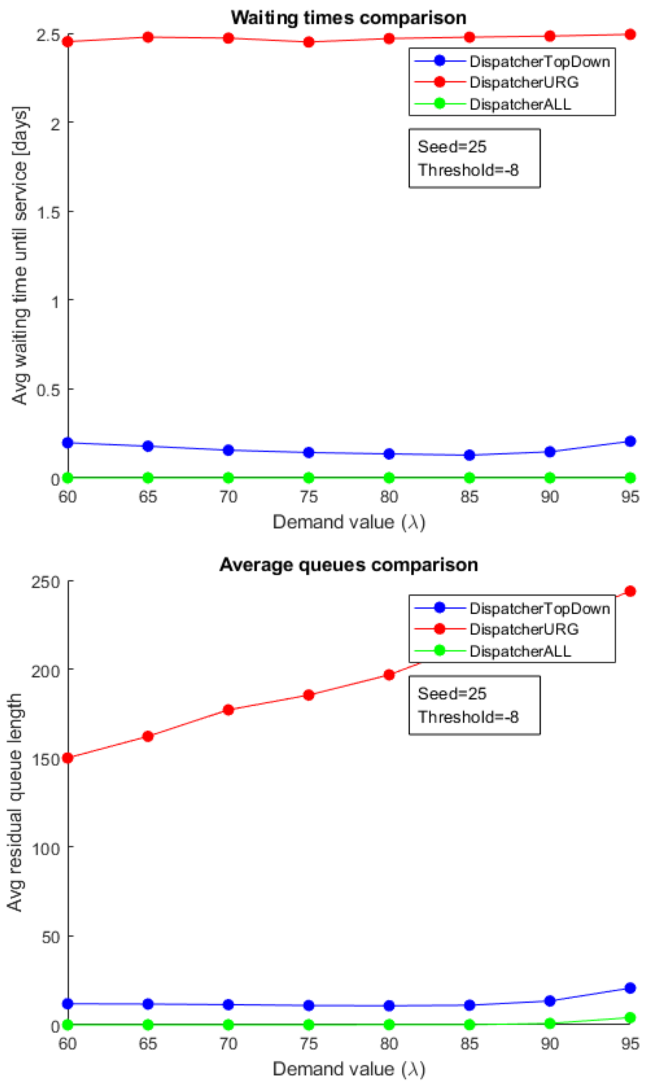


Figure 5.7: Waiting time and residual queue with varying demand

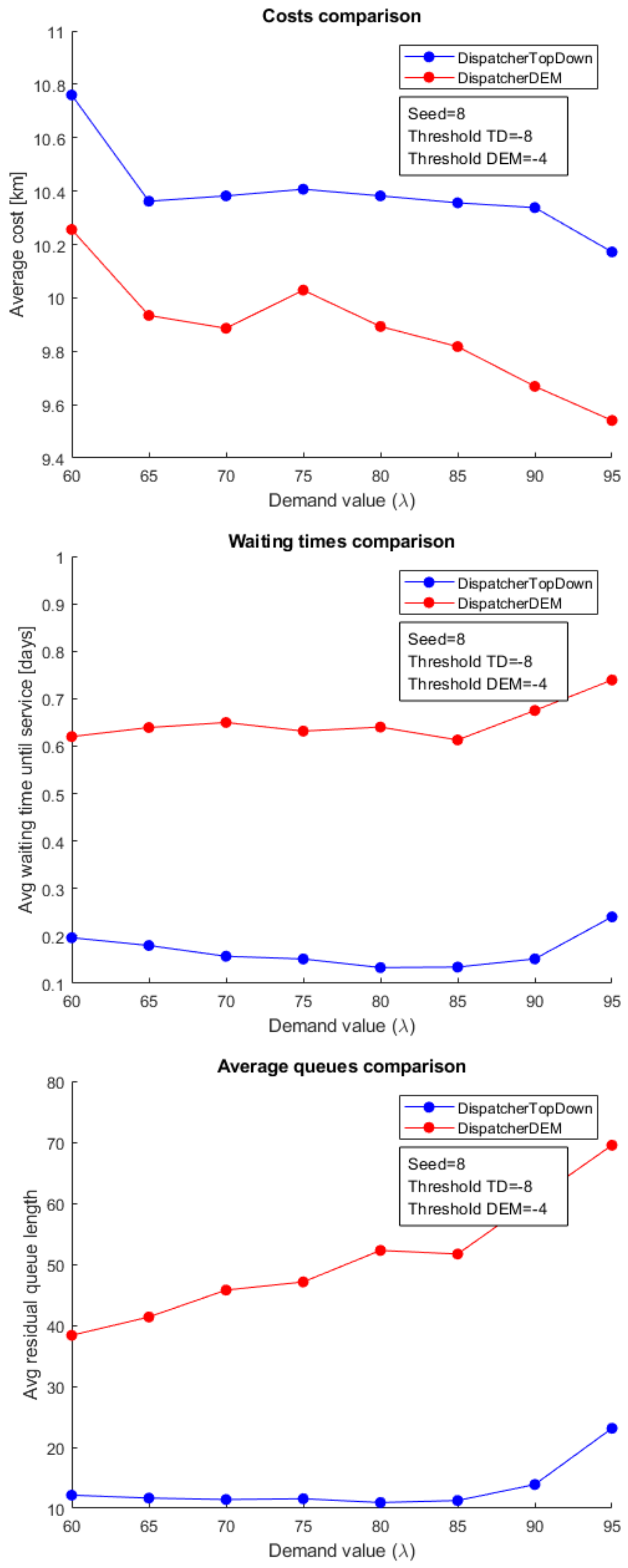


Figure 5.8: Performances comparisons (TopDown VS DEM)

5.3 Scenario 2

In this scenario we try to change the geography by considering a different customers' distribution. In particular we assumed that customers are uniformly distributed (each cell has the same probability) and that the depot coordinates are [105, 110] (in the center of the region). Figure 5.9 shows an example of the map of customers in a certain day. The black square represents the depot, the green dots are the customers chosen by the dispatcher, the red ones represent the urgent ones, and finally the black dots are the customers left for later.

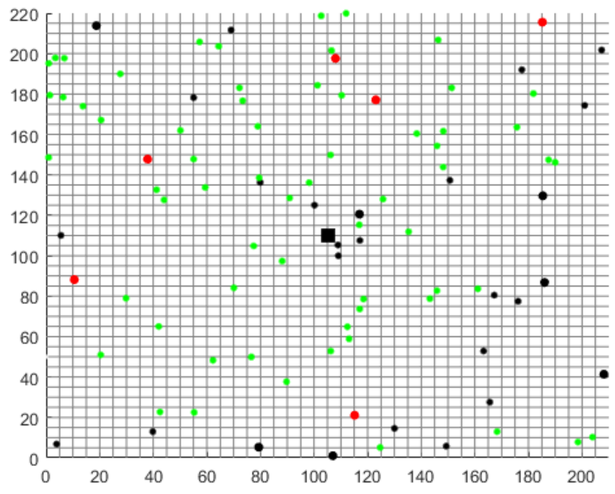


Figure 5.9: Customers' locations map.

5.3.1 Comparison with the benchmarks

Again, in order to analyze the performances of the $top - down$ approach in this different context, we compare the results with those ones obtained using the *dummy* strategies.
As before, we perform a sensitivity analysis in order to choose the right cut-off threshold. The analysis is analogous to the one described in section 5.2.1, so I won't go further into the details. The optimal threshold after the analysis turns out to be -20.

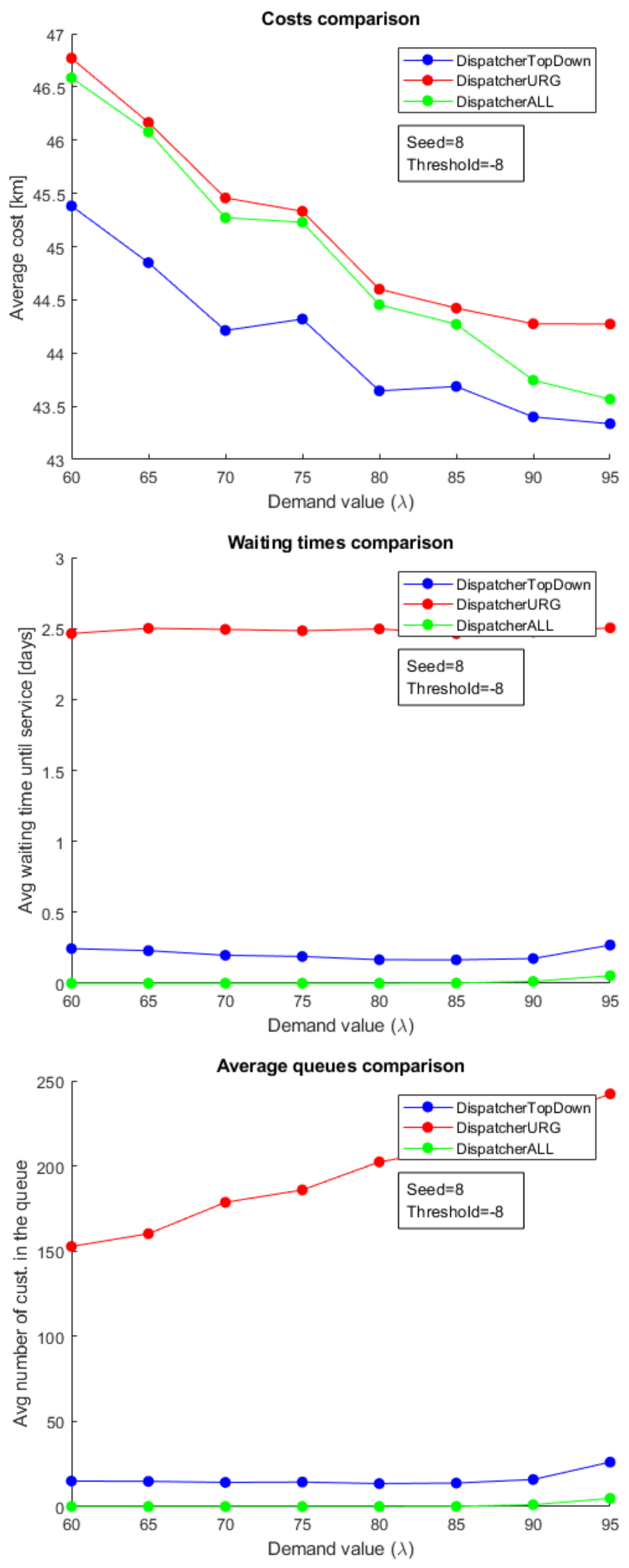


Figure 5.10: Performances comparisons (TopDown VS benchmarks)

Afterwards, as in the preceding scenario, we analyze the behaviour of average cost per customer, average customers' waiting time in the queue, average length of the residual queue at the end of the day. The results are shown in figure 5.10 and they are similar to those obtained in scenario 1. The main difference is that in this case the average costs are higher, as customers are farther from the depot.
In figure 5.11, again, we have the saving using *DispatcherTopDown* instead of *DispatcherURG* in terms of percentage (computed as in scenario 1). In this case the savings seems to be lower than in the other case, maybe because in a more regular situation, the dummy strategies work better.

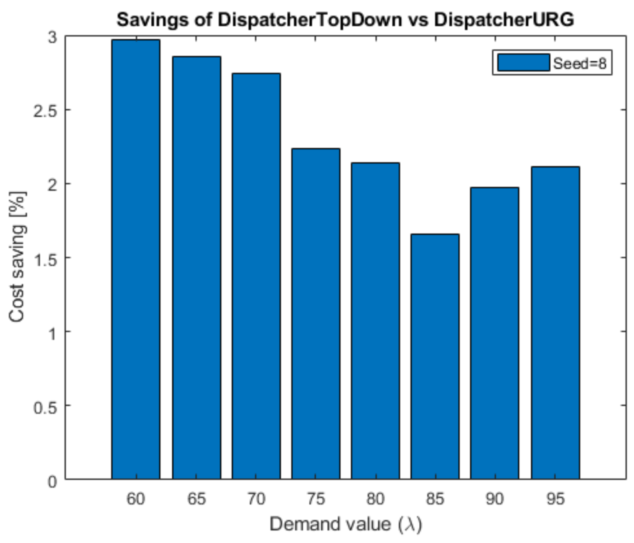


Figure 5.11: Percentage of saving (TopDown VS URG)

Conclusion

After having described the features of the vehicle routing problem (VRP), I focused on a particular real-world application involving a dynamic VRP, in which orders are revealed dynamically over time and it is necessary to develop a strategy to choose each day the right pool of customers to serve.
The proposed algorithm consists of three phases: the *learning*, whose aim is to develop a metric for the following phase based on the extra mileage criterion, the *dispatching*, which consists in the choice of the customers to serve, and finally the *routing*, through which customers are allocated to different routes.
We decided to use a top-down approach to choose the customers, based on a preliminary step in which all the customers in the queue are divided into clusters, followed by a phase in which the service of the least promising customers is postponed for a later day.
The performance of our strategy are compared to those obtained using two simpler strategies (serving all the customers and serving only the urgent ones) in two different scenarios. The result of the analysis is that there is a saving in the cost (average number of km traveled to reach a customer) up to the 6%.
The savings aren't as high as we expected, and the performances of the algorithm are not totally satisfactory. One of the possible factors that contributed to this result is the optimization model used to cluster the customers in the dispatching phase. Probably the clusters built through this model are not sufficiently good, and this affects the following choice of customers.
One possible solution could be building the routes through the router, already in the dispatching phase. This solution implies that the C++ code of the router must be modified in order to include in the output the routes and not only the total routing cost.
Instead of using a fixed cut-off threshold, we could use one that varies accord-

ing to some features of the customer considered, in order not to penalize the customers located near the depot, that don't bring enough benefit in being chosen.
The algorithm could be extended in order to take into account the possible rejection from the customer to the proposed delivery date and the dimension of the orders.
Another possible next step could be to consider the delivery times, that in the case of a furniture distributor, are extremely relevant (sometimes building a furniture is really time-consuming).
Despite the numerous limitations, the proposed approach is undoubtedly a good starting point for the development of more sophisticated algorithms.

Bibliography

[1] J. Bramel and D. Simchi-Levi, *A Location-Based Heuristic for General Routing Problems*, Working Paper, Department of Industrial Engineering and Operations Research, Columbia University, 1992.

[2] P. Brandimarte, G. Zotteri, *Introduction to Distribution Logistics*, New York: Wiley, 2007.

[3] G. Clarke and J. Wright, *Scheduling of vehicles from a central depot to a number of delivery points*, Operations Research, 12(4):568-581, 1964.

[4] G. B. Dantzig and J. H. Ramser, *The Truck Dispatching Problem*, Management Science, 6(1):80-91, 1959.

[5] J. Desrosiers,Y. Dumas, M.M. Solomon, F. Soumis, *Time Constrained Routing and Scheduling*, M. O. Ball, T. L. Magnati, C. L. Monma, and G. L. Nemhauser (eds), Handbooks in Operation Research and Management Science: Network Routing, 8:35-139, 1995.

[6] S. Dolan, *The challenges of last mile logistics & delivery technology solutions*, businessinsider.com, May 10, 2018, web, August 9, 2018, <https://www.businessinsider.com/last-mile-delivery-shipping-explained?IR=T>.

[7] M. Dorigo, *Optimization, Learning and Natural Algorithms*, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[8] M.L. Fisher and R. Jaikumar, *A Generalized Assignment Heuristic for Vehicle Routing*, Networks, 11:109–124, 1981.

[9] A. Fraser, *Simulation of genetic systems by automatic digital computers. I. Introduction*, Aust. J. Biol. Sci. 10:484–491, 1957.

[10] B.E. Gillett and L.R. Miller, *A Heuristic Algorithm for the Vehicle-Dispatch Problem*, Operations Research, 22:340–349, 1974.

[11] F. Glover, *Ejection chains, reference structures and alternating path methods for traveling salesman problems*, Discrete Applied Mathematics, 65(1-3):223-253, 1996.

[12] F. Glover, *Future paths for integer programming and links to artificial intelligence*, 13(5):533-549, 1986.

[13] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[14] C. Groër, B. Golden, E. Wasil, *A library of local search heuristics for the vehicle routing problem*, Mathematical Programming Computation, 2:79-101, 2010.

[15] B. Kallehauge B, J. Larsen, O.B. Madsen, M.M. Solomon, *Vehicle Routing Problem with Time Windows*, 2005. In: G. Desaulniers, J. Desrosiers, M.M. Solomon (eds) Column Generation. Springer, Boston, MA.

[16] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, Science, 220(4598):671-80, 1983.

[17] G. Laporte, F. Semet, *Classical heuristics for the capacitated VRP*, The vehicle routing problem, Society for Industrial and Applied Mathematics, Philadelphia, PA, 109-128, 2001.

[18] S. Lin, *Computer solutions of the traveling salesman problem*, 44:2245-2269, 1965.

[19] F. Patrone, *Metodi per la soluzione di problemi dinamici di routing di veicoli*, Master thesis, IV Facoltà di Ingegneria, Corso di Laurea in Ingegneria Gestionale, Politecnico di Torino, 2014.

[20] V. Pillac, M. Gendreau, C. Guéret, A. Medaglia, *A review of dynamic vehicle routing problems*, European Journal of Operational Research, Elsevier, 225(1):1-11, 2013.

[21] C. Rego, *Node-ejection chains for the vehicle routing problem: sequential and parallel algorithms*, Parallel Computing, 27(3), 201-222, 2001.

[22] P. Toth, D. Vigo, *An Overview of Vehicle Routing Problems*, The Vehicle Routing Problem, 1-26, 2002.

[23] R. W. Wolff, *Poisson Arrivals See Time Averages*, Operations Research, 30(2):223-231, 1982.

[24] Xin-She Yang, *Metaheuristic Optimization*, Scholarpedia, 6(8):11472, 2011.