Université catholique de louvain

LINGI2990
Travail de fin d'études

Optimizing the delivery of prepared meals

Student :
Guillaume Massin (3022-08-00)
Teacher :
Yves Deville
Readers :
(Jean-Baptiste Mairy)
François Aubry
Michael Saint-Guillain

17 août 2015

Abstract

Nowadays, optimization problems take a huge part of our industrial society. This may not be seen by a regular person's point of view but almost every company have to deal with productivity issues. There exists a lot of different kinds of optimizations problems that can be analysed. For example, one might want to maximize the number of products made during a day to maximize a production. Management entities might also want to attribute rooms to events or resources to several tasks that have to be performed to maximize their final profit.

This paper focuses on a specific problem dealing with food delivery services. Several approaches exist and have been found in the literature and will be briefly exposed here. However the main point of this paper will be to find the best way to apply local search on such a problem statically as well as dynamically.

This problem may seem easy at first sight but this is a real world problem with every difficulties that come with it. Indeed, each customer can register its order at a different time and the problem may vary from one minute to another.

We will here discuss on how to improve the efficiency of the planning given to the delivery man. We will explore local search solutions to try to reduce as much as we can the total distance travelled by the delivery man as well as the delivery latenesses.

Acknowledgements

This paper was not produced without effort and difficult time management.

First, I would like first to thank the professor Yves DEVILLE and his assistant Jean-Baptiste MAIRY for being patient with this work during the semester.

I would like to thank then my parents and my family for not losing their mind when I was losing mine.

Thanks to my fellow students of last master and university year that will hopefully graduate along with me.

I also want to thank the members of my kot because even if they made this year really emotionally and timely draining, they were of a good support during the long days (and nights) of work.

Last but not least, I would like to thank a lot the company which this master thesis was prepared with. Even tough their name can not be cited here, they were of a good help for defining the problem and giving an actual and concrete real world problem to work on.

Contents

1 Preamble **1**
- 1.1 Introduction . . . 2
  - 1.1.1 Content . . . 2
- 1.2 State of the art : definitions . . . 4
  - 1.2.1 Definition of Vehicle Routing Problem . . . 4
  - 1.2.2 Optimizing the objective . . . 4
  - 1.2.3 Capacited VRP . . . 6
  - 1.2.4 One-to-One problem . . . 7
  - 1.2.5 VRP with time windows . . . 8
  - 1.2.6 Static versus Dynamic . . . 8
  - 1.2.7 Definition of our problem . . . 9
  - 1.2.8 Adaptation of the objective . . . 10
- 1.3 State of the art : resolution . . . 11
  - 1.3.1 Global notions for resolution methods . . . 11
  - 1.3.2 Construction heuristics . . . 11
  - 1.3.3 Local search . . . 13
  - 1.3.4 Dynamic approaches . . . 16
  - 1.3.5 Other approaches . . . 17
  - 1.3.6 Stacker crane problem . . . 17
  - 1.3.7 Dynamic VRPPD . . . 19

2 Working with our problem **21**
- 2.1 Specifications . . . 22
  - 2.1.1 Specificities for our particular problem . . . 22
  - 2.1.2 Size of the orders . . . 22
  - 2.1.3 Company objective . . . 22
  - 2.1.4 Other suppositions . . . 23
- 2.2 Data . . . 24
  - 2.2.1 Description of the data . . . 24
  - 2.2.2 Exploiting the data . . . 25
  - 2.2.3 Suppositions on the data . . . 25
  - 2.2.4 No management of the orders processing . . . 25
- 2.3 A word on the algorithm . . . 26
  - 2.3.1 Common implementation . . . 26
  - 2.3.2 Global view on the algorithm . . . 26
- 2.4 Approaches that have been tested . . . 30
  - 2.4.1 Initial solution and order insertion . . . 30
  - 2.4.2 Possible moves in the local search . . . 30
  - 2.4.3 Ejection chain, another approach . . . 35
  - 2.4.4 Complexities . . . 37
  - 2.4.5 Meta-heuristics . . . 37
  - 2.4.6 Adaptive memory . . . 38

2.4.7 Impose a minimal number of orders . . . 39

3 Presentation of the results **40**
3.1 Results . . . 41
3.1.1 Static problem . . . 41
3.1.2 Dynamic problem . . . 52
3.1.3 Conclusion from the data . . . 52
3.1.4 Results without meta-heuristics . . . 52
3.1.5 Adaptive memory . . . 58
3.1.6 Imposing a quota of orders . . . 59
3.2 Comparison to official solution . . . 60
3.3 More improvements . . . 61
3.3.1 Multi-orders delivery . . . 61
3.3.2 Concern about the worst lateness . . . 61

4 Conclusion **62**
4.1 Conclusion . . . 63
4.1.1 Context . . . 63
4.1.2 First analysis . . . 63
4.1.3 Final results . . . 63

Bibliography **64**

5 Appendices **i**
5.1 Graphs for the initial solutions . . . i
5.2 Graphs for the tabu influence for day 1 for all search methods . . . iii
5.3 Graphs for the tabu influence for the modified searches 3 to 6 . . . viii
5.4 Graphs for the static objective evolutions . . . x
5.5 Graphs of comparison to the company solution . . . xiv

Chapter 1

Preamble

1.1 Introduction

Optimization problems are a key part of our modern world. Unfortunately, if a company does not find an operation, a service or a sector relevant enough, it might be cut off or not explored at all. What would happen if the delivery planning of medicines was so badly produced that providing the hospitals and pharmacists were not worth the investment ? This kind of issue is called *Vehicle Routing Problem (VRP)* and is the class of problem we will explore here, but for a way less heavy situation.

A very well known example of VRP is the Travelling Salesman Problem. A salesman wants to visit a set of cities one time each and wants to follow the path with the minimum distance. The salesman starts from a city and goes back to the same one at the end. This problem is similar yet different to the one stated above.

The problem we will be interested about in this paper is a problem of delivering food to paying customers. This paper was done in collaboration of a real existing company but that we won't be able to cite due to a non-disclosure agreement. We will take the case of this company desiring to make partnerships with restaurants that don't have their own delivery service. They want to offer the restaurants to allow some customers to order meals by them and handle of the deliveries at home. Customers would actually order their meals to the company and it will itself contact the restaurants to handle the rest of the operations. The objective will thus to minimize the time a customer will have to wait to get its meal and the lateness of such a delivery service.

The company wishes to pay the delivery man by commission and thus would like, for their employee satisfaction, that they have at least each three deliveries to make every day.

This problem may seem easy at first sight but this is a real world problem with every difficulty that comes with it. Indeed, each customer can register its order at a different time. We thus have to deal with an incomplete problem at every second. At the beginning of the day, we have no actual idea of how good our temporary solution is as long as we don't have the information about the next orders. Also, every decision must be done quickly and efficiently because the delivery men cannot wait several hours to be told which order to handle, which would results in great lateness for the deliveries.

A lot of different approaches exist for solving such problems and there exists quite a lot of papers on the subject and some of them will be (briefly) cited and explored here. But it was decided beforehand that this paper would focus on local search resolution. Furthermore, it was decided that the resolutions would not take a statistical dimension and would work "blindly" without having a further knowledge on statistics and probability distribution of the orders, leaving faith in the randomness of the real world.

This master thesis will aim at providing an answer and a good solution to this particular problem while exploring the possibilities that exists in that domain. We will discuss on how to improve the efficiency of the planning given to the delivery man and try to reduce the total distance travelled by the delivery man and to reduce the delivery lateness as much as we can.

1.1.1 Content

This paper will have several chapters.

In the first one, we will start by explaining a formal definition of what a *Vehicle Routing Problem* is and how it applies to our actual problem. We will then expose different techniques that can be explored.

After that, we will explain how the problem was considered and specified to allow us to work on it and briefly explain what kind of data was available for us to work on to be able to perform our analysis and reach conclusions. A small word will be told on how the prototype was conceived to justify the legitimacy of our results and conclusion even tough the data not being publicly available, you will have to have some faith in our results. We will show several local search algorithms that we have tested and explain how they work.

Hopefully, we will then present the results we were able to obtain from the experiments on the prototype and try to make some conclusion out of them.

Finally, before a final conclusion on the work and the experiments done, some improvement ideas will be proposed for a potential modification of the problem that could lead to efficiency and productivity improvement for the company but that we were not able to cover here because they were out of the scope of this paper.

1.2 State of the art : definitions

This problem enters in the category of *Vehicle Routing Problem*, but with some specificities. This section will explain what is a general *Vehicle Routing Problem* and the elements that are necessary to define it. A lot of papers already give definitions of the VRP in the literature. For example, the thesis referenced as [7] is pretty complete on the subject. In this section, we will develop the definition of a VRP, some of its variants and finally explain how exactly is characterized our problem.

1.2.1 Definition of Vehicle Routing Problem

Rough definition

Vehicle routing (most commonly called VRP's) is a class of combinatorial optimization problems. Its first appearance was in the paper of George Dantzig and John Ramser [3] where the author wanted to optimize the deliveries of gasoline. In such problems, we dispose of some good (or service) as well as geographical locations where we have to deliver (or perform) them. Disposing of a fleet of a certain number of vehicles, we have to distribute the different locations where there are actions to perform between the elements of this fleet while trying to maximize an objective depending of the situation.

Elements of the problem

To be able to define the VRP in a precise way, we have to introduce several notions :

- A *customer* corresponds to a geographical location where the service has to be performed or the good has to be delivered. Depending on the necessity of the problem, all of them may or may not need to be visited.
- A *vehicle* is the entity travelling trough the geographical area from the depot to the customers to perform the deliveries or the services. Most of the time, the vehicles show some specific features such as a transport capacity, a speed limit or other limitations.
- A *depot* is the geographical location from which a vehicle starts and ends its travelling. There may be several depots or only one depending on the problem definition.
- The *road network* represents all the routes and paths that the vehicles can use. Usually, this network is represented by a graph. On this graph, every node is a customer or a depot and the vertex linking them represents the paths used by the vehicles to go from one to another. Each vertex is associated to a cost, usually corresponding to the distance separating the locations.
- *Routes* are the sequences of locations that a vehicle will visit one after another. They usually start and end at a depot and list all the customers that the vehicle visits in the right order.
- The *fleet* is the ensemble of all the available vehicles for the problem. The vehicles in the fleet don't necessary need to be homogeneous and the vehicles can present different characteristics one from another.

1.2.2 Optimizing the objective

With all those elements, the problem is to maximize or minimize an *objective* associated to the problem definition. More notions will be necessary :

- The *objective* of the VRP is the minimization or the maximization of a value. The most usual objective is to minimize the total distance travelled by the fleet of vehicles. In some other cases, we may need to consider also (or only) the time at which the vehicles arrive at the locations.

- A *solution* (S) of the VRP is an assignment of customers to the vehicles and thus their assignation to the routes. The solution may either be complete if all the customers are visited by some vehicle or incomplete if not. Each solution is evaluable with a function computing the objective.
- Some *constraints* are usually necessary to define the specificities of a VRP. Those constraints apply to the solution of the problem and impose restrictions often inherent to the problem definition. They can consist of properties associated to the vehicles, the ordering of the customer locations and so on.

To the objective is associated an objective function that can be defined, in the general VRP, as the sum of the distances from all the routes :

$$\text{minimize } f(S) = \Sigma distances_k$$

A solution S_{opt} will be called optimal if it is better than every other possible solutions :

- $\forall S : f(S_{opt}) \leq f(S)$

Formal definition

The problem is defined as a simple complete weighted graph : $G = (V, E)$. Let us consider a day where a number n of customers uses the service. $V = 0, ..., n$ is the set of vertices. The depot is denoted to be the vertex 0 and the next n vertices correspond each to one customer location. E is the set of edges, arcs, linking the vertices between them. Each vertex can show incoming and outgoing arcs, denoted respectively $\delta_i^+ and \delta_i^-$. Each edge $(i, j) \in E$ has an associated cost, a weight. In the basic problem, c_{ij} is simply the distance between the two geographical locations that represent i and j. It thus corresponds to the cost of choosing this edge as a part of the solution. For some of the problem, the distances are considered to be symmetric : going from i to j or j to i has the same cost, so $c_{ij} = c_{ji} \forall i, j \in V$. They also have to respect the triangle inequality : $c_{ij} \leq c_{ip} + c_{pj} \forall i, j, p \in V$. So going from i to j directly should cost less (or the same) than going via p.

Notations

To help in our definition, we will fix some notations :

- e_i is the customer i, the action i to be performed (delivery or service)
- R is the set of routes (and $|R|$ the number of routes)
- r is a route and r_i is the i^{th} route
- $r.set = e_1, ...e_n$ is the set of the customers in a route (not necessarily in the right order)
- $r.seq = (e_1, ...e_n)$ is the sequence of the visited customers in the right order
- $r.seq(i) = r(i)$ is the element at position i on the route
- $r.seq[e] = r[e]$ is the position of the customer e in the route
- $|r|$ is the number of customers on the route r
- K is the set of vehicles (and $|K|$ the number of vehicles)
- k is a vehicle and k_i is the i^{th} vehicle
- $dist(r)$ is the total distance of the route
- $first(r)$ and $last(r)$ are respectively the first and last element of the route

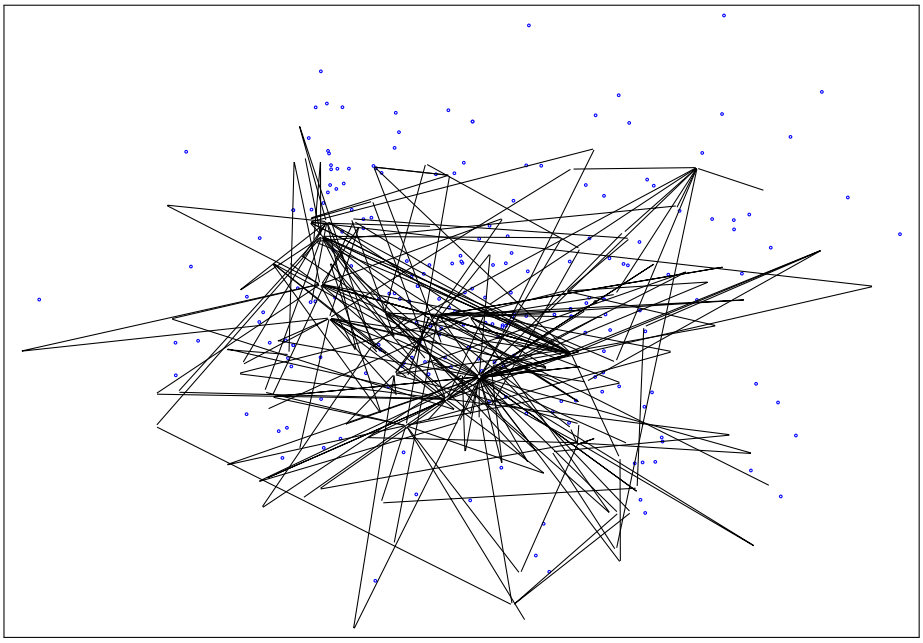

Figure 1.1: Graph corresponding to an instance of the problem

- $first(r)$ and $last(r)$ are the depot
- $|S|$ is the number of vehicles assigned to a solution

By definition, a route starts and ends at the depot. A route can thus be represented as a sequence of vertices $r.seq = (first(r), ..., last(r))$ corresponding to the sequence of locations that the vehicle will visit one after the other. For example, $r(2)$ corresponds to the customer that will be visited in second by the vehicle associated to this route. The first and last element will always be the vertex 0, the depot so $first(r) = last(r) = 0$. If the sequence contains the segment (e_i, e_j), it means that the the edge (i, j) is part of this route and that its cost has to be taken into account.

The figure 1.1 shows an example of solution and representation for a given instance of the problem.

Each vertex (each customer) is associated to one and only one route respecting the constraints :

- $|S| \leq |K|$: the solution doesn't use more vehicles than the number available.
- $\cup_{r_i \in S} = V \backslash \{0\}$: each customer is visited.
- $\cap_{r_i \in S} r_i.set = \varnothing$: a customer is visited no more than once.
- Distance(solution) $= \Sigma_{r \in S}(\Sigma_{i=0}^{|r|} c_{r(i)r(i+1)})$

A solution is feasible if it respects the three first constraints. If not, it is called infeasible. The last constraint is only helpful to compute the objective function and the objective value. The problem of finding a solution of such problem is NP-hard.

1.2.3 Capacited VRP

As stated, some constraints of the VRP can impose limits on the vehicles. For example, they can have a limited capacity. A variant of the VRP with such constraints is called the *Capacited Vehicle Routing Problem*. For example, one might have a limited number of orders that it can carry. Or for the execution of services, it can have a quota of a maximum number of actions to be performed at

the end of the day. We thus have to introduce the notions of capacity of the vehicles and demand of the customers.

We have K homogeneous vehicles with the same capacity Q. To the other notations, we thus add :

- Q is the capacity of the vehicles (and thus the routes)
- q_i is the demand associated to the customer i
- $demand(r) = \Sigma_{i \in r.set} q_i$ is the accumulated demand on the route

We thus have to add the capacity constraint to the ones stated above :

- $\forall r \in R : |r| \geq 3$

This constraint also has to be respected for the solution to be feasible.

1.2.4 One-to-One problem

In fact, the definition we gave above is not enough to define the kind of problem we are facing. Here we, we will show some other classification that can be done for similar problems. In other problem definitions, each customer might be paired with a source to pick-up the goods. This kind of problem is called one-to-one pick-up and delivery problem.

Besides the vehicle capacities, the problem is thus more specific as the vehicles might not dispose of all the goods to deliver directly at the depot. They might have to charge and discharge several times during their trip.

The one-to-one pickup and delivery problem can be split into three classes of problems :

- Vehicle routing problem with pick up and deliveries (VRPPD): if a vehicle can serve more than one request at a time.
- Stacker crane problem (SCP): if a vehicle can only perform one request at a time.
- Dial-a-ride problem (DARP): if the goods to be transported are actually people.

We won't develop the third case since it doesn't fit with the problem we want to analyse here.

Pick up and delivery

In this variant of the problem, we don't only have to deal with the delivery of the orders to the customers but we have to pick them up at a pick-up locations before being able to deliver them. If a vehicle can carry goods for several customers at once, this is called pick-up and delivery.

The capacity constraint of a vehicle is thus not handled the same way since its loading will very trough time.

In this problem, we don't only have customers as delivery locations but also pick-up ones. We thus have $V = 0...n + m$ vertices, n being the number of customers where deliveries have to be done and m the number of pick-up locations that can be used.

The ordering of the order is also important and we can define :

- p_i is the pick-up location of the order i
- d_i is the delivery location of the order i
- $load(k)$ is the load of a vehicle

And we have the additional constraints :

- p_i and d_i are executed on the same route
- $r[p_i] < r[d_i]$, the pick up of i occurs before the its delivery.

And the capacity constraint has to be adapted :

- At any moment : $load(k) \leq Q$ (the loading of a vehicle cannot surpass its capacity)

Stacker crane problem

The variation of the VRP called stacker crane problem follows the same principle as the pick-up and delivery variant besides the fact that every vehicle has a unitary capacity constraint.

Besides, the capacity of the vehicle is only of one order. We thus have the constraint :

- $Q = 1$
- $r[p_i] = r[d_i] - 1$ the pick up of i occurs right before the delivery (and vice-versa).

Every order can be thus considered as a single "special vertex" in the graph consisting of the pick-up vertex, the delivery vertex and the edge linking them. Indeed, this special vertex has to be found as it is in the final solution.

1.2.5 VRP with time windows

With that kind of problem, the vehicles can often not perform their tour in any order. Most of the time, there is a specified time at which the customers have to be visited and the orders have to be picked up and delivered. Each location has thus an ideal time of reaching from which the vehicle has to try to be as close as possible. With that ideal time is defined a time window $[e_v, l_v]$ where e_v and l_v are respectively the lower and upper bound associated to the vertex. A service can not start before its lower bound. It is thus called a *hard* constraint. On the opposite, it can end after the upper bound even tough it is not wished (it is a *soft* constraint).

With those new parameters, a new notion can be developed : the lateness on a route.

$$\text{Lateness} = \forall r \in R : \Sigma_{i=0}^{|r|} max(0, \text{Time of delivery}_i - \text{Wanted time of delivery}_i)$$

We want to reduce this lateness as much (if not more) as the distance. It will thus be added to the objective function to be part of the minimization with a factor α determining its importance.

$$minimize f(Sol) = \Sigma distances_k + \alpha \Sigma latenesses_k$$

1.2.6 Static versus Dynamic

The definitions of the *Vehicle Routing Problem* presented before is general. However, when we want to deal with real world problems, all the information of the final solution is not pre-emptively known. This is called a dynamic problem. On the opposite, in the static approach, orders are all known in advance. In the real world, data can arrive at any time, which can considerably change the necessary approach. It is thus important to consider specific approaches for the static and the dynamic version of the problem.

In the dynamic version of the problem, a route associated to a delivery man can be modified any time as soon as a new information arrive. That being said, a local good solution with half of the data might lead to an awful solution once all the data is known.

Compared to the static problem, most of the aspects of the problem remain actually the same :

- We dispose a of fleet of vehicle
- We know the travelling time between two points
- We allocate orders (and their positions) into the routes associated to a delivery men
- We have an objective to optimize

What changes is that the orders are not known from the start but once they are done during the day.

1.2.7 Definition of our problem

Modifications from the general problems

We will now talk a bit more about our problem. As stated before, the main idea is to deliver meals ordered from restaurants to customer locations. It has been determined that a delivery man can only carry one order at once and have to deliver it right away. Our problem is thus actually a stacker crane problem. If we allowed to have bigger bags for the vehicles, allowing vehicles to handle multiple orders delivery at once, it would be a VRPPD. However, we chose to still consider the possibilities and look in for work that have been done in the literature for this kind of problem as well. Besides, maybe later in the future, the company might want to allow their delivery man to so. So if a method that has been proven to work well for the VRPPD is efficient in our case also, we might still want to explore it.

If the delivery men cannot carry several orders at once, the problem can also be considered as a one-to-many-to-one problem with vehicles of unlimited capacity and vertices constituted in fact of a pair (restaurant, customer). But this possibility has been less explored in the literature and is not the approach we will take here so we didn't develop it.

As we have a really concrete problem, we can state several particularities from the base problem. Further in the paper, some analysis will be done on what happens when changing some of them. Our additional specificities require to add more constraints to the problem.

- There are two types of vehicles, disposing each of one out of two types of bags : big or small. A vehicle with a big bag can carry a small order but a small bag cannot contain a big order.
- Each order is associated to a pick-up and a delivery location.
- The delivery locations are the customer locations and the pick-up locations are the restaurants.
- The pick-up and the delivery of an order has to be performed on the same route by the same vehicle.
- The vehicles can carry only one order at once. Their capacity is thus only one and the delivery has to be done right after the pick-up. (Stacker crane problem)

This problem thus have all the elements and constraints of the SCP but with some adaptation. The fleet is not homogeneous. Instead of $|K|$ vehicles with capacity Q, we dispose of $|K_2|$ vehicles of capacity Q_1 and $|K_2|$ vehicles of capacity Q_2, corresponding to big and small bags. Some constraints thus have to be adapted.

Those constraints are untouched :

- $\cup_{r_i \in S} = V \backslash \{0\}$: each customer is visited.
- $\cap_{r_i \in S} r_i.set = \varnothing$: a customer is visited no more than once.
- Distance(solution) = $\Sigma_{r \in S}(\Sigma_{i=0}^{|r|} c_{r(i)r_{(i+1)}})$
- p_i and d_i are executed on the same route
- $r(p_i) < r(d_i)$, the pick up of i occurs before the its delivery.
- $r(p_i) = r(d_i) - 1$ the pick up of i occurs right before the delivery (and vice-versa).

But those constraints are modified :

- $|S| \leq |K_1| + |K_2|$: the solution doesn't use more vehicles than the number available.
- $q_i = Q_1$ or Q_2 : the size of an order is either small or big
- A small vehicle (k_2) doesn't carry a big order (capacity Q_1).

If we are taking care of the supplementary objective that every vehicle has to do at least three deliveries, we can add the constraint :

- $\forall r \in R : |r| \geq 3$

1.2.8 Adaptation of the objective

The problem have *time windows*. This implies that the actual objective doesn't consist in only minimizing the distance but also other parameters. As the orders correspond to food and meals, we can suppose that they have a specific ideal time of delivery. If a customer wants a meal for lunch, he doesn't want to be delivered at 4 pm. According to that, we will add the total lateness to the objective function.

One other time variable that can be taken into account for a productivity issue is the time of the vehicles being inactive. Indeed, the meals also have time at which they are ready. It is a waste of time if a vehicle arrives early at a restaurant or a customer and has to wait some time before being able to go back on its way. We will call this variable the overtime.

Another thing that could have been into account is the time at which the delivery men perform their last delivery. As we have no information on what is the official closing time of the service and as this value is really closely linked to the lateness of the orders, we let this parameter aside. A delivery man finishing really late is either due to a order having to be delivered late or due to huge global lateness in all the order deliveries, otherwise it would be assigned to another delivery man.

- Lateness = $\forall r \in R : \Sigma_{i=0}^{|r|} max(0, \text{Time of delivery}_i - \text{Wanted time of delivery}_i)$
- Overtime = $\forall r \in R : \Sigma_{i=0}^{|r|} max(0, \text{Wanted time of deliveryy}_i - \text{Time of delivery}_i)$

There is also a time window at the depot defining the earliest possible start and the latest come back of a vehicle.

The adapted objective is thus :

$$minimize f(Sol) = \Sigma distances_k + \alpha \Sigma latenesses_k + \beta \Sigma overtime_k$$

As the lateness and the distance are a way bigger concern than the overtime, we will set the parameters to $\alpha = 0.1$ and $\beta = 0.01$.

1.3 State of the art : resolution

Finding a solution for such problems requires thus to apply optimization procedures. Optimizing the solution can be a complete or incomplete procedure. Complete procedures guarantee to find an optimal solution where incomplete procedures don't. As we work with real world data in real time, we probably won't be able to afford doing complete strategies.

1.3.1 Global notions for resolution methods

A lot of approaches have been developed over the years to solve VRP's. Before explaining them in details, we will explain the differences between a *constructive* and a *perturbative* approach.

Constructive approach A technique is constructive if it starts from an incomplete solution and adds more details to it to finally reach a complete solution. It then builds a partial solution from another partial solution. **Perturbative approach** A perturbative approach will start from a solution and change it. It will move from a state to another state in its neighbourhood depending on the possible moves that can be done (which will be defined by the problem and the search heuristic).

1.3.2 Construction heuristics

Before being able to apply local search techniques, we have to dispose of an initial solution. Several construction heuristics are well known in the literature. These methods are *incomplete* and *constructive*. They will try to construct a set of routes by adding customers one after another until a complete solution is found. At each step, we only consider the current situation and not the past ones. With those heuristics, we will try to obtain a solution of a good quality without having to spend much time and resources. Three methods are well known and widely used.

Saving heuristics

Using this approach, the algorithm starts by putting each customer on a route on its own and then starts merging the routes in a greedy way.

For every pair of node (i,j), a saving value is computed : $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. This value corresponds to the gain that would occur by merging two routes, one ending at i and the other starting at j.

If there exists at least two routes and merging those two routes is possible without exceeding the maximum demand (if a CVRP) on this route, they are merged. Mathematically, if $\exists last(r_1) = i$ and $first(r_2) = j$ and $demand(r_a)+demand(r_b) < Q$ then they are merged and $Sol = (Sol \backslash \{r_a, r_b\} \cup r'$ with $r' = r_a x r_b$ being the new route replacing the two others.

As long as there is a possibility to merge, the algorithm continues to do so. The final solution could show more vehicles than available and thus would not be feasible in the case of a CVRP. It would be the role of a next step in the optimization to change that. If we are dealing with a SCP, like here, the problem can continue until the right number of routes is obtained. This heuristic is illustrated on the figure 1.2.

Insertion heuristics

With those heuristics, we start from the right amount of routes and we add customers one after another into them. A route is qualified as *open* if we can potentially add a new customer to it. It is considered *closed* otherwise. At the start, no customer is assigned to any route. At each step, the heuristic decides which customer is to be added and at which position in which route.

For example, a simple case would be if there is an single open route at once and the heuristic chooses the customer to insert as the closest one from the last position on that route. Once there is no more customer that can be added to the route (because of a capacity constraint or another parameter), the current route is closed and another one is opened.

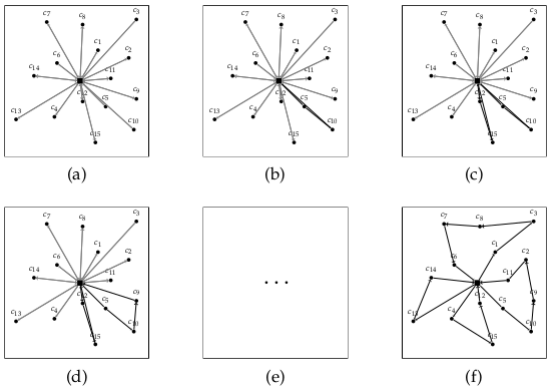

Figure 1.2: Illustration of the saving heuristic [7]

Other versions might be a lot more complicated and consider several spots on several routes. Several routes are then open simultaneously. For example, one can try all the possibilities and check the gain of adding each customer at the end of each route (or even at each position between the existing orders) and select the best insertion from this criterion. The figure 1.3 illustrates the behaviour of this heuristic.

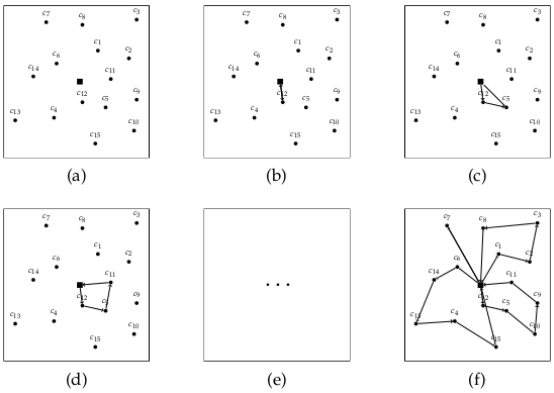

Figure 1.3: Illustration of the insertion heuristic [7]

Sweep heuristic

The sweep heuristic is based on clusterisation. It means that it groups the customers with a predefined criterion. This one works in two phases. First, it separates the customers in several sets based on their geographical positions or another criterion. Then in the second phase, it builds one route for each cluster, that route visiting all the customers of the cluster. In facts, it can be seen as resolving the travelling salesman problem for each cluster.

The most commonly used and easy to represent criterion for the clustering is the geographical position. It is called sweeping heuristics because the algorithm will literally sweep trough the customers and select them. The algorithm will take the depot as the center and, let's say, start from the east to create an empty cluster. It will then rotate around the center. Each newly found customer will be checked to see if it fits in the cluster. Otherwise, a new cluster, and thus a new route, will be created and so on. For example, for a trying to add the customer v in a CVRP : if $demand(current\_cluster) + q_v \leq Q$, it is added to the cluster. Once all the customers have been processed, the clusters will be analysed to generate the best routes (and order sorting) for them

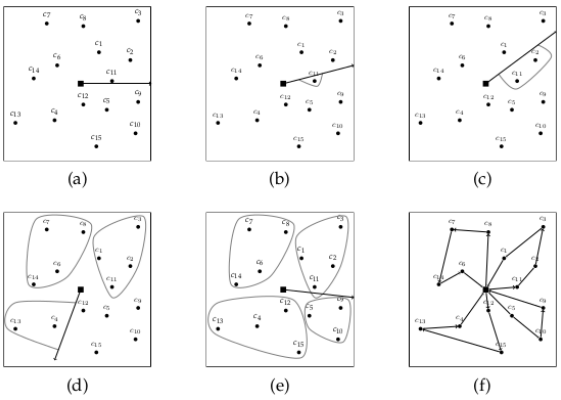

Figure 1.4: Illustration of the sweep heuristic [7]

specifically.

1.3.3 Local search

In this section, we will explain what will be the core of this paper.

When we have a solution, we can perform operations on them to try to improve them. This is the idea of the local search. It starts from a solution and jumps to another one (feasible of not), this is an *incomplete* and *perturbative* method. When applying the local search, each solution is considerer individually. The *solution space* is defined as the set containing all the possible solutions (being infeasible or feasible). The algorithm will jump from one solution to another by going trough the neighbourhood of the current solution which is defined as all the states accessible from the current solution by applying some specific operation.

Neighbourhoods can be very different depending on which neighbourhood operator is applied. The neighbourhood of a solution S corresponding to the application of the operator *op* will be noted as $N_{op}(S)$. The operator can also take parameters to specify which features of the problems are to be taken into account. The size of the resulting neighbourhood can really differ related to the complexity of the operator that is used.

At each step, the quality of the current solution is evaluated. The search will move through the solution space trying to reach a better solution. Each step is thus a perturbation of the current solution.

The search will keep track of the best objective value observed so far and its associated solution. This solution is called h *incumbent* solution. At each iteration, the search will possibly check the feasibility of the current solution and, if it is better than he best solution computed so far, record it as the incumbent solution.

Operations on the routes

During the search, several operations can be done on the route to find the possible neighbours.

1. Inserting a customer to a route : the insertion of a customer to a route is done by adding the customer i to the route r. The new route is thus $r'.set = r.set \cup \{i\}$. We can define a position to add the customer at a specific place p in the sequence. (insert(i,p,r))
2. Removing a customer from a route : the new route is the same without the selected customer, the other ones are untouched $r'.set = r.set \backslash \{i\}$: (remove(i,r))
3. Merging two routes, denoted $r = r_1 x r_2$. Thus $r.set = r_1.set \cup r_2.set$ and the sequences are just joined one after another.
4. Swapping customers is actually the combination of the two first moves.

First improvement of best improvement

There are several possible strategies to select the next solution among the neighbourhood. We can either select the first move that improves the solution (*first improvement*) or the one among all the possible moves that improves the solution the best (*best improvement).*

In the first method, the gain associated to a neighbourhood is evaluated immediately during its construction. As soon as one improving move is found, the search jumps to it and goes to the next step.

With the second strategy, all the neighbourhood has to be constructed at each iteration. It is thus longer to compute but tends to find better improvements at each step.

A variant of the best improvement strategy is to select the best move among a set of the x best ones, instead of selecting automatically the best one. The reason of why it is important to use such a strategy is explained in the next section (1.3.3).

Intensification and diversification

As said before, a strategy for the travelling trough the neighbourhood would be to not select the best move but one among the set of defined size containing the best ones. To understand the usefulness of such a move, we have to define the concepts of *local optimum* illustrated on the figure 1.5.

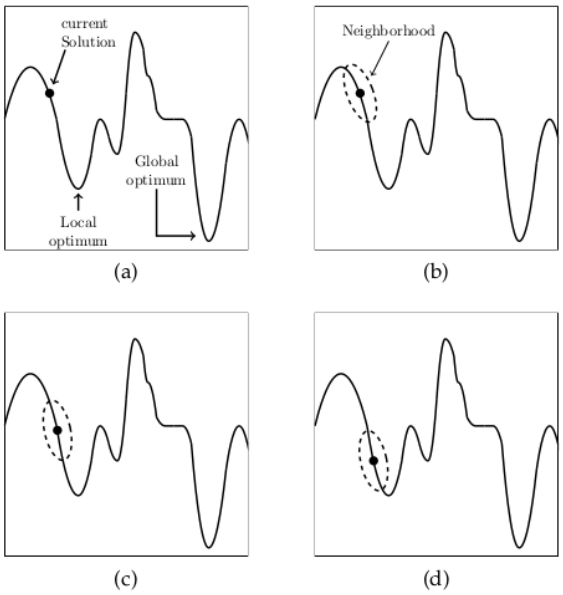

Figure 1.5: Illustration of local optimum [7]

Intuitively, the main aspect of the search is to go trough the solutions one after another and improve it iteratively until the best one is found. However, by doing so, we are not sure to reach the optimal solution, or even to be close to it. Indeed, the search being limited by the neighbourhood selection, some states might not be reachable following the direct route.

As illustrated on the figure 1.5, we can even reach a state where there is no solution in the neighbourhood that improves the objective value but where the final accessible solution is still not that good. This is called a *local optimum.* By focusing on the most promising search directions, we might be doomed to reach one sooner or later. If we are lucky, it is the good and near the global optimum but most of the time, the chances are low.

This first step of focusing on going to better solutions is called *intensification* and obviously needs a complementary concept to achieve better quality : this is the *diversification.* By the application of other heuristics, we can force the search to explore other areas and be able to

eventually jump out of the local minimum. This approach can stop the search to be cyclic and stay in the local pit.

To be able to reach several solutions with better chances, randomness will be an important factor. It will allow to not systematically follow the same path when considering the same current solution, which would lead to useless loop of operations. Sometimes, the algorithm could also have to restart, trying to find a new solution from scratch. Thanks to the randomness, we will most likely get other solutions.

Beside, a restart doesn't necessary need to be build completely from scratch but can also use information from a previous solution to accelerate itself and be guided more quickly to a good solution.

Meta-heuristics

Some improvement can still be made when regarding the selection of the neighbourhood. By adding some meta-heuristics, we can try to be more clever in the direction we want to guide the search. A meta-heuristic is a higher level procedure used to help the program to improve its solution without being the main aspect of the search. Most of the time, meta-heuristics will help to escape local optimum and lead the search closer to an acceptable solution. Here are some well known meta-heuristics :

1. **Tabu search** is very useful to avoid being stuck in local optima. The idea is to forbid a move when a similar move has already been performed. To do so, an additional structure will be made : the tabu list.

   Depending on the problem structure, this list will contain either moves, customers or other elements characterizing the solution. Each item of the list is linked to a move that has been performed and that is now forbidden for a fixed number of iterations. Therefore, the tabu list has a fixed size which is the number of steps of the search the information will be kept. At each step of the search, when a neighbour is selected (and thus a move performed), the tabu information is stored in the list. If the list is already at its maximum capacity, one of its elements is removed, most of the time the oldest one.

   Each time a new neighbour is to be considered, it is compared to the criterion stored in the tabu list and can be ignored if some of its elements correspond to the ones stored in the list. If the neighbour is allowed and selected, the search continues normally and the tabu list is updated as explained earlier.

   A tabu neighbour can still be checked before being rejected to see if the move improves the *incumbent* solution. In that case, we might allow the search to overpass the tabu criterion.

2. **Reactive tabu search** : the capacity of the tabu list might vary during the problem to change the balance between *intensification* and *diversification*. One typical way to do it is to increase the size of the list when the quality of the solution improves (to promote a promising search area) and to decrease it when the solution quality decreases (forbidding more moves, promoting diversification).

3. **Simulated annealing** is based on the thermodynamic principle of annealing in metallurgy. [6] It is a probability based meta-heuristic where a new solution may or may not be selected in the neighbourhood depending on if this solution is better than the current one and if we didn't find a new solution for a long time. Typically, when we have a current solution S and a new potential solution S', we will compute $\Delta = f(S) - f(S')$ where $f(S)$ is the objective value associated to those solutions. The probability of selecting S' as a new solution is :

$$P_{\tau_i}(S'|S) = \begin{cases} 1 & \text{if } \Delta \geq 0 \\ e^{(\Delta/\tau_i)} & \text{otherwise} \end{cases}$$

The τ_i parameter starts at an initial value and will be updated during the search decreasing along with the number of iterations performed without finding a new solution. ($\tau_{i+1} = \alpha\tau_i$ with typically $0 \leq \alpha \leq 1$). When a better objective value is found, τ is set back to its original value. This update occurs after a defined number of iterations proportional to the size of the neighbourhood.

It will allow to not move to some states if they are too close from the current solution and are not improving it. This will tend to exit local pits and jump to another place in the search.

4. **Variable neighbourhood search** uses several neighbourhoods instead of one. They are usually sorted by their size. When at a step to choose the new solution among the neighbourhood, it will check if this decision improves the current solution. If so, then this neighbour is selected and the search goes to its next step. If not, the next step of the search will choose the next neighbour among the next neighbourhood (if there is one).

1.3.4 Dynamic approaches

It is probably good to recall now that the actual problem we are interested in is the dynamic version of the problem. To consider a dynamic problem, two approaches can be adopted : [2].

- Each new event (such as the arrival of a new order) produces a new static problem. Each of those static problems only contains events that have been made until this one. There is no forecast on the future orders. The problem is thus solved with the actual data so far without prediction. Those incomplete solutions might need some readjustment for them to be still feasible in the complete problem. One huge drawback of this approach is that rebuilding a new static problem at each step might consume a lot of resources to attain a good level of optimization again. It is then not suitable for real-time goals.

- The static problem is applied only once at the beginning with the available data at the start so that an initial solution is obtained. Each time a new event arises, the solution is updated with heuristic methods (insertion, move, etc.) before further optimization. The orders are thus added one after another in the current solution and re-optimized further. This is generally the chosen approach for real-time problems. Between the different events, a more robust algorithm runs so that the data can still be optimized.

Waiting strategies

In our definition of the dynamic problem, we have stated that we would not do any prediction and stochastic analysis of the future when dealing with the actual current orders. There is still a need to anticipate the future in the behaviour of the vehicles. A way of anticipating the future requests is to apply different waiting strategies : what does a vehicle do when there is no order to perform at the moment ? Some basic strategies can be used.

- Wait-First: when a vehicle has performed a delivery, it waits at the location of the last customer until another order is assigned to it.

- Drive-First: when a delivery is performed, the vehicle goes back to the depot or a "high interest zone" defined externally by the manager of the program.

A combination of the two strategies can also give better results such as divide the different geographical zones into segments and use the drive-first behaviour until a segment border is to be crossed. The vehicle then waits until further order.

A buffering strategy could also be used consisting in holding the next orders for a while before assigning them to a vehicle. The vehicles can then move to an area where their next destination is likely to be picked.

1.3.5 Other approaches

The point of this paper is to analyse the local search performances. We have made our choice on this particular method. However, other approaches exist in the literature and can be good to explore in other circumstances. We will show some of them here without going into details. As our aim is to work with real world data and real time analysis, we will prefer incomplete search procedure. Indeed, it is very unlikely that we will be able to find the optimal solution of the problem in a decent amount of time. Furthermore, the problem being very dynamic (many orders can come any time), this reinforces the idea that achieving optimal solution is unrealistic. A lot of possible approaches have been exposed in the literature to solve the dynamic SCP and VRPPD[2]. The following sections will briefly present some of them to give you alternative from the approach chosen here.

1.3.6 Stacker crane problem

If we consider the problem as stated, allowing only one order at once by vehichle, it is a stacker crane problem. As said before, we talk about a "stacker crane problem" when resources have to be directly delivered to their location after the pick-up. This restriction is generally due to vehicle capacities. This subsection will briefly expose some already explored resolution algorithm for this kind of problems.

Powell (1987) [11]

The approach from this paper presents the following characteristics :

- Divide the planning area into regions.
- We have to know the probability distribution or future requests.
- The problem is represented as a network flow where each node represents a region that needs to be served at a specific time.
- There are two types of arcs : known or empty and stochastic.

This choice is not possible for us since we do not have the statistical data and do not plan on doing stochastic previsions.

Yang et al. (1998) [17]

- No need for stochastic data
- Each request has a time window
- The static problem is represented as a integer linear program
- It is solved many times in a rolling-horizon framework

This study tells that this solution outperforms other ones but requires high computational efforts. This high cost is a big drawback that we can't afford for our actual problem.

Yang et al. (2004) [18]

This one is an update of the previous one where :

- Requests can be rejected
- Deliveries can occur outside the time windows incurring a penalty

Compared to the previous one, there are two distinct optimizing policies :

- Optimizing the MIP of the static problem every time a new request arrives
- Same as the previous one but assumes some knowledge about the probability distribution of incoming requests

This one seems too complicated since we don't want to handle the rejection of the orders. This aspect of the problem is not done by the local search. Besides, once again, we don't deal with probabilities about the new requests.

Tjokroamidjojo et al. (2006) [16]

This approach tries to quantify the value of knowing the requests in advance.

- Consider a fixed value for the length of the horizon
- No stochastic information is considered
- Solving the MIP every time a new request arrives
- The different policies are characterized by (τ, T) where
  - τ represents the time in advance the requests are known
  - T represents the time in advance the assignment are made

This study shows that policies (3,1), (5,3), etc. shows better results than the ones like (1,1), (3,3), (5,5). Indeed, it thus shows that knowing the orders way before having to deliver them is beneficial. This results will help us in deciding the way of assigning to orders but doesn't give us any lead to a resolution.

Mes et al. (2007) [8]

This approach is different from the others as it is agent-based.

- The time window constraints are *soft*
- There are several types of agents with their own goals
  - Vehicle agent
  - Fleet agent
  - Job agent
  - Shipper agent
- Every time a request arrives, the job agent makes the vehicle agent bid with a price, an expected arrival and departure time
- The job agent chooses the highest bidder but with the price of the second highest

This approach is also quite specific and is really far from the original plan of making analysis of a local search optimization.

Gutensch-wager et al. (2004) [6]

Tabu search and simulated annealing heuristic

- Solving the static problem several times without stochastic knowledge
- Objectives adapted to the on-line algorithm

This approach seems more realistic regarding the type of problem we have. The analysis is done first on the static problem and then, with the conclusions that have been made, adaptations are made to fit the dynamic approach. This is the kind of approach we will use here.

1.3.7 Dynamic VRPPD

Here, vehicles can serve more than one request at a time. This can be a useful extension to the basic problem and it could be interesting to study the gain that this improvement would allow. The vehicles would have a capacity of Q units and can perform as many orders simultaneously as their capacity allows them.

Although this problem will not be analysed here, some approaches might still be interesting now or later if the company wants to allow this feature.

Feuerstein and Stougie (2001) [4]

This paper shows two possible approaches :

- Minimization of the time of the last destination served (makespan)
- Minimization of the sum of completion times (latency)

Simple algorithm called DLT (Don't listen while travelling)

- Once the vehicle is at the origin at time t, it follows an optimal tour including all requests that have not been served yet and goes back to the origin
- Objectives adapted to the on-line algorithm

Ascheuer et al. (2000) [1]

This paper shows three on-line algorithms :

- Replan : when a new request is obtained, the old request is performed and then an new optimal tour is computed containing the new request
- Ignore : same as DLT
- Smartstart

Smartstart is the only on that can handle vehicles with $Q > 1$.

Savelsbergh and Sol (1998) [15]

This optimization captures constraints about time windows, vehicle capacities and working period restrictions. This study is originally for a problem on several days where we want to minimize the total driver's pay (depending on the travelling distance and the number of nights away from home). It is more complex than the problem we are interested in. The global decomposition of tasks here is :

- Decomposition into several static problems with a subset of requests on a rolling horizon framework
- Each static problem is solved with a brand-and-price based heuristic

Mitrovic-Minic and Laporte (2004) [10]

The objective is to serve all the requests while minimizing the total distance. To do so, there is a two phases heuristic :

- Cheapest insertion procedure
- Tabu search algorithm

Different waiting strategies have to be considered to be efficient.

Mitrovic-Minic et al. (2004) [9]

This paper shows an extension of the previous one but with a double-horizon objective

- Short term objective minimizing the routes
- Long term objective minimizing the slack time to insert new requests

Gendreau et al. (2006) [5]

This paper shows great possibilities regarding local search on VRPPD.

- Tabu search algorithm
- Neighbourhood based on ejection chains
- Adaptive memory to maintain a good quality of solution
- Quick first solution with a simple insertion procedure to every solution in the adaptive memory
- Fast local search on the better solution for improvement
- As long as there is no new request or no new completion, the local search keeps running
- Master-slave procedure to speed up the tabu search

This paper talks about the VRPPD but is actually interesting since the approach is similar to the one we would like to use.

Sáez et al. (2008) [14]

Adaptive predictive protocol for capacitated dynamic VRPPD

- No time window
- Objective function takes into account waiting and travel time for the requests
- Objective takes into account future requests using probabilities based on historical data

This one is left apart by default since it also uses historical data to make statistical assumptions, which we don't have.

Chapter 2

Working with our problem

2.1 Specifications

2.1.1 Specificities for our particular problem

As said in the previous chapter, our problem shows some specificities compared to the general VRP : it is a stacked crane problem. As the problem is very well defined by real world means, some other features have to be fixed.

Here are some assumption on the management of the orders and their assignation to the moving vehicles.

We do not consider diversion of the vehicles: a vehicle can not diverge from its destination once in movement, which means :

- All the vehicles start and end their routes at a common depot.
- The assignment of an order to a delivery man is only definitive if he is already on its way to the corresponding vertex. This means that as long as the pick-up of an order has not started, it can be assigned to another vehicle.
- By extension, this means that when a vehicle is in movement, its destination is fixed.
- The order currently being processed is unmovable from one delivery man to another, as well as all the orders that have already been delivered (the routes in the algorithm have to remain consistent with the routes that have been travelled by the vehicles so far)
- A new order can be given definitively to a delivery man only if it is currently waiting.

In this problem, we also not consider :

- Anticipation of future requests stochastically : there is no analysis on where could come the next order regarding the previous distribution of customers
- Time dependant travelling times : the distances are considered to be as the crow flies. The actual geographical routes are not known. The distances are therefore ideal and do not take into account the different detours that could be performed in real life.

2.1.2 Size of the orders

Let's still remember that orders can be of two different sizes : big or small. The vehicles have the corresponding size of bag. On vehicle with a small bag can thus not carry an order of a big size.

2.1.3 Company objective

As repeated earlier, the objective of a VRP is to optimize an objective function. Here, the standard objective is to minimize several parameters. Their order of importance is :

1. Maximize the customer satisfaction (reducing the lateness of the orders)
2. Minimize the route distance of the vehicles
3. Minimize the slacking time of the vehicles to maximize productivity
4. Minimize the cost of operation
5. Pay the delivery man equally as much as possible

In this paper, we consider that the most important objective to achieve for that kind of delivery service is the customer satisfaction, which is directly linked to the delivery lateness. This is not idyllic, but as we have a limited number of vehicles, big lateness could arise. The company would try to minimize the lateness to, on one side, make the clients wanting to use the service again and, on the other side, be able to serve a maximum amount of customers.

As this objective is obviously linked to the total distance since the bigger the distance, the longer the travelling time. It is logical that it is taken into account in the objective.

The third parameter specifies that we don't want vehicles to arrive too early at the locations because it will have to wait before being able to continue its route. It's thus a waste of time. This parameter is actually less important than the other ones in the objective function because we can assume that it is directly linked to the two others. Besides, one way to reduce this slaking time is just to wait before assigning a delivery to the vehicle and thus keeping it inactive if an order is not urgent. This is actually done in the dynamic problem since orders are not assigned to a delivery man as soon as they arrive (this will be explained in the section covering the algorithm and its implementation 2.3).

The fourth objective is not measured and actually implicit. We obviously want the algorithm to give good results but also to run fast. Indeed, this is a real time problem, we can not afford to have a good solution by having to wait two hours before being able to deliver an order. The company would go bankrupt.

The last objective is directly inherent to the wish of the company to also satisfy its employees as much as its own profit. Indeed, the delivery men are paid regarding to the number of deliveries they perform, but they are also paid for at least three deliveries, even if they don't perform any. The company would like to pay them for actual deliveries and not doing nothing.

2.1.4 Other suppositions

There are other assumptions that are made on the orders management.

No cancellation

For example, in the data, their is no entry about orders cancellation. We assume that when an order is made, it is definitive and thus not handle the case of when a customer would change its mind of ordering the meal.

Time windows at the depot

Also, the time windows can be flexible. The starting time of the vehicle has been set to 11pm as the standard time before any customer could want to receive an order. The max arrival time however has not been set and is considered to be dependant of the orders themselves. As this program doesn't handle the decision of accepting an order or not, it is considered that if the order has been accepted by the manager, it will have to be delivered regardless of the returning time of the vehicle to the depot. The upper limit of the return of a vehicle should not be abusive if the algorithm is good.

Infinitely small time windows

The aim is to reduce the lateness of the delivery at the customer locations. To do so, the time windows will considered to be infinitely small. The hard lower bound is the ideal time of the delivery and no order can be delivered before that. The soft upper bound is also this ideal time and every minute above is considered as lateness. This is an express request from the company. As the delivery time is defined by the user and checked with an external source, we considered that if this time has been accepted, the algorithm has to try to achieve it.

Waiting strategies

The main waiting strategy that is used here is the one asking the vehicles to wait at their current location before performing another pick-up. As long as the time is not close enough to the order pick-up time, it is not definitely assigned. A few minutes before the pick-up time, the order is assigned to the delivery man that has to handle it in the best solution so far. The strategy is thus a wait-first strategy with order buffering (see section 1.3.4).

2.2 Data

2.2.1 Description of the data

To do our analysis, we have have received a set of actual real world data from the company. Because they are under a non-disclosure agreement, they are not redistributed and available with this paper. For the same reasons, they will not be explained in details. However, we will explain here which elements were needed to do our analysis and how we have used them. The given data were covering a period of 19 days, we could thus dispose of 19 different instances of set of orders, deliveries and the assignation that had been done by the company on those days. None of the data has been or will be used for other purpose than doing researches and experiments for this work and no personal data has been extracted for personal interest.

Beside the data on the orders, we also disposed of data on the actual tours the vehicles were assigned to on each day. From this data, we could compare solutions obtained here to the solutions actually performed by the company.

From the data, we could extract four kind of in information :

Order assignations to delivery men

The first information was about the different assignations of the orders among the delivery men and their actual travels. This information was mostly useful for comparison purposes.

1. ID : the ID of the assignation of the order to a vehicle
2. Food order ID : the id of the order in this association
3. vehicle ID : the id of the delivery man in this association
4. Desired date of start : the desired date from which the delivery man starts moving to pick up the order at the restaurant
5. Actual date of start : the actual date from which the delivery man starts moving to pick up at the restaurant
6. Desired date of pick up : the desired date for the pick up at the restaurant
7. Actual date of pick up : the actual date for the pick up at the restaurant
8. Desired date of delivery : the desired date for the delivery at the customer
9. Actual date of delivery : the actual date for the delivery at the customer

Order information

The second type of information we had was about the orders. Obviously, we wouldn't be able to work on the problem if we didn't have those information. In those data were also contained all the delivery locations.

1. ID : the id of the order
2. Customer ID : the id of the customer involved in the order
3. Restaurant ID : the id of the restaurant involved in the order
4. Date of creation : the creation date of the order
5. Date of desired delivery : the desired delivery date of the order

6. Date at which the order has to be ready at the restaurant : the hour we expect the meal to be ready at the restaurant
7. Size of the order : the size of the bag associated to the order (small or big)
8. Coordinates : the polar coordinates of the location where to deliver the order

Delivery man positions

The next batch of data concerns the delivery man and their routes in the real solution. This was not the most exploited one. The main use of this set was to know if the vehicles had a small or big capacity by associating them with their orders' size.

1. ID : the ID of the entry
2. Vehicle ID : the delivery man ID
3. Coordinates : the x and y polar coordinate
4. Date of the entry : the date at which the position was recorded
5. Accuracy : an estimation of the precision of the position

Restaurant information The last set of data was about the different restaurants that the company was working with. It was obviously important to know them as they represent all the pick-up points. In this paper, no name will be cited.

1. ID : The id of the restaurant
2. Coordinates : The latitude and longitude of the restaurant

2.2.2 Exploiting the data

Several structures have been used to extract the data. Basically, the first step has been to extract roughly all the data and make them fit into personal structures containing all the information from every file.

One of the first things to do on the data was to check the orders and their associated delivery man to retrieve the capacity of their bags, which was not directly stated.

2.2.3 Suppositions on the data

Some suppositions have been done on the data so that the problem is better defined and easier to exploit.

1. The lateness at the delivery determine the total lateness of an order

2.2.4 No management of the orders processing

It was repeated that the objective of this paper is to analyse the performances of route assessment with a defined set of orders using local search. The aim is thus not to create a program capable of handling a real delivery service itself. That being said, the program does not handle any aspect of the order processing by the customer or checking if the addition of that order to the problem prevents its feasibility or doesn't impose a huge lateness. This aspect of the problem is considered to be handled by another source (let them be another program or the human hand). An order in the data has to be performed no matter what with all the impact it could have on quality the solution.

2.3 A word on the algorithm

In this section we will give a word on what we have done in our prototype to legitimize and explain how we got the results we will be able to give to show how we have obtained them.

2.3.1 Common implementation

We will first briefly talk about how our prototype works. As said before, the prototype runs on data instances. We can thus select which day we want to work on. The first step of the program is to select, for the selected day, all the orders that are to be performed that day.

The main function have several parameters that can be modified at first. Other parameters can be modified as well :

- The number of the day we want to perform the experiment on
- The number of loops (or amount of time) we want the local search to run
- The value α parameter in the objective formula
- The value β parameter in the objective formula

To ease the utilisation and experiments, the mode can be selected directly.

- 0 : Retrieve solution as performed by the company
- 1 : Perform the algorithm on a static instance
- 2 : Perform the algorithm on a dynamic instance

A lot of structures have been used.

- Personal structures representing orders, restaurants, vehicles, etc.
- A hashmap associating the id of an order to its other information
- A hashmap associating the id of a delivery man to its capacity
- A linkedList that will be the tabu list
- Structures containing the copy the incumbent solution
- Other structures taking a part for other roles of the algorithms

The roads are kept in another structure : *successors* (here another HashMap). For each entry in this map, the associated value is the ID of the next order on the road. This map contains orders ID as well as delivery men ID. To be sure to differentiate them, all the ID for the vehicles have been set to negative.

To retrieve a road, we thus start by fetching the value corresponding to a vehicle and get the next value, and so on, until the fetched value is again the vehicle value.

2.3.2 Global view on the algorithm

To have a better understanding of what we do exactly, we will develop here the big lines of the algorithm we will use.

Static problem

For the static problem, there is not many specificities to state. All the orders are known from the start and added to the routes using an the insertion heuristic (see section 2.4.1 after). Once every order has been placed, they are shuffled applying the local search to try to achieve the best possible disposition.

```
General algorithm for the static version:
Initialize all the data from the data files
Add empty routes to the problem
Perform the initial solution (insertion)
Set the initial solution as the incumbent solution
Apply the local search
Return the incumbent solution
```

Dynamic problem

Unlike the static problem, a full initial solution containing all the orders can not be done. Instead, the orders are being added one by one on the empty routes and blocked one after another. The dynamic problem thus needs some additional structure.

- A structure keeping tracks of the orders that have been added to the problem
- A structure keeping tracks of the orders that have already been assigned to a delivery man
- A priority queue playing the role of the simulator

The simulation of the dynamic events didn't recreate a static from scratch each time an order was added. Instead, it was done using an event priority queue. For the dynamic processing, only two kinds of events are important :

1. Creation of an order : a customer places an order and it has to be added to the list of orders processed by the algorithm.
2. Assignation of an order to a vehicle : the time has come for an order to be definitively assigned to a delivery man.

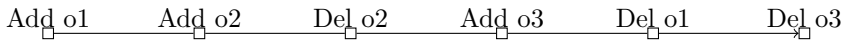

Figure 2.1: Illustration of an event queue

The raw data has been processed to check the time of those two kinds of events. Those are the two only essential parameters to be taken into account.

When an order is placed, it has to be added to the order pool. The addition will be done the same way as the insertion heuristic (again, see section 2.4.1). It is now inserted in the problem and can be considered as well in the local search.

When an order has to be picked-up, it is definitely assigned to a vehicle. It then has to be removed from the order pool and be made unmoveable.

To be consistent with the real world problem, the time of assignation to the delivery man will correspond to the time of its departure from its previous delivery location. In the computation of the objective value, the total time and the lateness, this time will be taken into account to be as realistic as possible. This also means that as long as an order is not assigned, the computation of the objective value of a route is more optimistic since the vehicle potentially has more time to reach the locations. In fact, part of this time will be spent waiting.

The local search is performed an amount of time proportional to the time between two events to be as close as possible of a real world case where the problem can be optimized as long as there is no new information to add to it.

When all the events have been processed, the program can provide the final solution for the day we were working on.

The global algorithm is thus a little different here :

```
General algorithm for the dynamic version:
Initialize all the data from the data files
Add empty routes to the problem
Set the initial empty solution as the incumbent solution
Create and fill the event queue
For every event in the event queue
  If the event is an order creation
    Retrieve the incumbent solution
    Add the order to the problem
    Updates the incumbent solution (with now contains the new order)
    Apply local search for a time proportional to the time before the next event
  If the event is an assignation
    Retrieve the incumbent solution
    Block the order
    Apply local search for a time proportional to the time before the next event
Return the incumbent solution
```

The blocking of an order is done using three structures : one containing all the ID of all orders that have been assigned so far, one containing their assignation to the routes (another successor structure) and one containing their assignation times. To block the order, there is a limit case to handle where, on a route, all the orders preceding the order to be assigned are not assigned as well. This case occurs only when the routes are poorly optimized but still should be handled.

```
Block order :
If one of the previous orders on the road is not blocked (limit case)
  Block them
Add the order to the list of blocked order
Add the order in the successor structure containing only blocked orders
Set the time of assignation of the order
```

Applying the local search

In the previous algorithms, we stated that we were using local search. This actually is a loop containing several steps. Each time a local search loop is performed, the program applies the right neighbourhood exploration according to the move we want to try. Each move results in its own operation but eventually results in a new successor structure from which we will compute the new objective value.

The next algorithm explains how the local search is applied in the static version of the algorithm

```
Apply the local search :
While a number of iteration (or time spend) is not reached
  Explore the current neighbourhood according the the moves tried
  Select the best neighbour
  Move to the selected neighbour
  If the new solution is better than the incumbent solution
    update the incumbent solution with the new solution
```

The selection of the best neighbour is done applying the moves explained in the next section 2.4.2.

Meta-heuritics

The applications of the meta-heuristics are more specific and will be explained in the next section where the operations are explained in more details.

Distance between the locations

The distance between two locations is computed as the crow flies with a simple function taking the X and Y polar coordinates of the two locations. This is most standard way. It was provided by the company to be sure that our results are compatible.

Computing the objective values

The objective value is computed from the successor structure. A function go trough the successors for the route and compute the lateness, overtime and distance by looking at the data and the list of successors. All the times are expressed in seconds. The lateness is only computed as the lateness at the customer locations. For the dynamic version, the time of assignation has also to be taken into account.

```
Compute objective for one route :
Start from the depot
Set the current time to starting time of the day
Set the distance, lateness and overtime to 0
While the next position is not the depot again
  If time of assignation is bigger than the current time
    current time = time of assignation
  Add the distance and time to reach the restaurant
  If time of pick-up > time of arrival
    Wait for the pick-up
  Add the distance and time to reach the customer
  If time of delivery > time of arrival
    Add to overtime
    Wait for the delivery
  Else
    Add to lateness
Return distance, lateness, overtime
```

To compute the complete objective, we simply have to compute and add the objective of each route.

Testing the prototype

Some unit tests have been performed to check the correctness of the main methods such as the route additions, computation and so on.

Methods has also been done to check the structure of solution and its feasibleness. Those methods check if there is no order missing in the solution, if they have all been processed, etc.

Some tests have been implemented are :

- Check if all the sizes of the orders are compatible with their delivery man assignation
- Check if every route has an end point corresponding to its delivery man ID
- Check if every order before an already delivered order (or in progress of delivery) is delivered as well
- Check, for the dynamic problem, that the delivered orders don't change their assignation (the vehicle and their order of delivery)

2.4 Approaches that have been tested

Now that we have defined our problem and the global algorithm, we will develop the operations in more details. Of course, for every of those operations, the algorithms won't add an order of big size to the route of a vehicle disposing of a small bag. We have worked in two phases. First we proposed and analysed our different algorithms and then we studied the impact of imposing a quota on the number of orders.

2.4.1 Initial solution and order insertion

For the static problem, the initial solution is created using an insertion heuristic. As the main constraint is defined by the time and not the geographical position, we let sweep heuristic aside. Indeed, it wouldn't help much to cluster the orders by location as they are not necessary to be delivered at compatible times. Also, merging the routes with the saving heuristic could have lead to good solutions if the merging was done regarding to the slacking time until the number of routes at the end was the right one. This would have been done by merging routes trying to spread the orders equally but this approach wouldn't necessarily be as good as the next one. And, as obviously the original solutions will be a lot worse than the one got after applying the local search, time has not been spend trying to find the best construction heuristic. The solution will thus be got in a more greedy way.

The program will start with empty roads with only vehicles starting at the depot : one road for one vehicle. On the static problem, orders are pre-sorted according to their arrival time to try to get closer of the dynamic problem. As soon as we add an order to the problem, it will be placed at a place determined by the adopted strategy.

1. Place the order at the end the road on which the vehicle that was available first
2. Place the order at the place on any road where it generates the smallest objective value.

The first strategy was deployed at first in the project. Later, the second one took its place because it showed a real improvement (as exposed in the section 3.1.1 in the results chapter).

Little by little, all the orders will be assigned to a delivery man in a correct way giving an acceptable list of routes but not optimized at all.

On those initial solutions, we will apply some local search algorithm to modify the routes and try to find new ones still satisfying the problem but minimizing the objective value, thus maximizing the customer satisfaction. Several algorithms have been explored and detailed in the next section.

Analysis on the best insertion heuristic also helped to choose the strategy to adopt when inserting a order into the dynamic problem.

2.4.2 Possible moves in the local search

As we said, the main point of this paper is to explore local search possibilities and compare them. In this section, we show some possible moves resulting in different neighbourhoods for the search. The ones exposed here have been implemented and tested on the data we dispose of, both statically and dynamically.

The moves presented here represent the available neighbourhoods reachable from a current solution. Only one of the moves in the neighbourhood will be selected and, only if the new solution shows a better objective value, the solution will be kept as incumbent solution.

For each move, a tabu list has been used. The values stored in that list are specific to each move and are detailed for each of them.

1: Swap orders

This move consists in swapping two orders from their respective routes. We have to go trough all the orders assigned to all the routes and check the objective value of their swap compared to the

original objective value. To do so, we have to compute the objective value associated to the routes that are taking a part in this change. We cannot just compute the gain on the switch itself because a lateness on an order can impact the rest of the route.

Obviously, symmetries can be observed on this type of algorithm. Swapping an order from road a to b, or the opposite leads to the same results. We thus only have to test swapping an order from a road with smaller index to a bigger index.

```
For every pair of route
  For every pair of order from the two routes
    Computes the value of the swap
    If the value is the best gain found so far
      Keep it
```

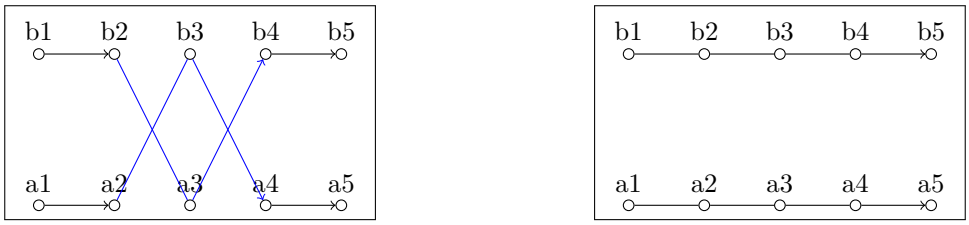

Figure 2.2: Illustration of the swap move

For this move, the tabu value is the ID of the two orders that were used in the swap. So at each iteration, two values enter the tabu list. This prevents the orders that have been swapped to be swapped again together but also with others to improve diversification.

The complexity of this method is quite high since it has to check all the possible swaps. Globally, we can see that there is a total number of n possible orders to swap with (n-1) other orders. For this algorithm, we thus obtain a time complexity of $O(n^2)$.

2: Remove worst order and add back somewhere

When looking at a route that seems to weight down the objective value a lot, it is sometimes due to an only order that is very badly placed. A huge gain might be obtained by removing this order from the route. This is the idea of this move : looking for the most interesting order to remove from a road. An order is considered interesting to remove if its removal from the road causes a big drop (and then a big gain) on the objective value.

There are thus two phases on this move :

1. Select one of the n orders to be removed from its route.

2. Try to add the order on every route, at every place. The spot giving the least augmentation on the objective value will be selected.

```
For every route
  Check the gain of removing the order
  If the gain is the best so far, keep that order in mind
Remove the worst order from its route
For every route
  Select the best new placement for the order that was removed
```

The complexity will thus have two phases too : O(n) to determine the worst order. Then O(n) to test every other possible placement. This gives us a final complexity of O(n+n) thus O(n).

In fact, there are two variants of the search. One allowing the addition of the removed order back on its original route and on assuring that it will be added on another route. The two methods are interesting to analyse since preventing the addition back to the same route promotes more

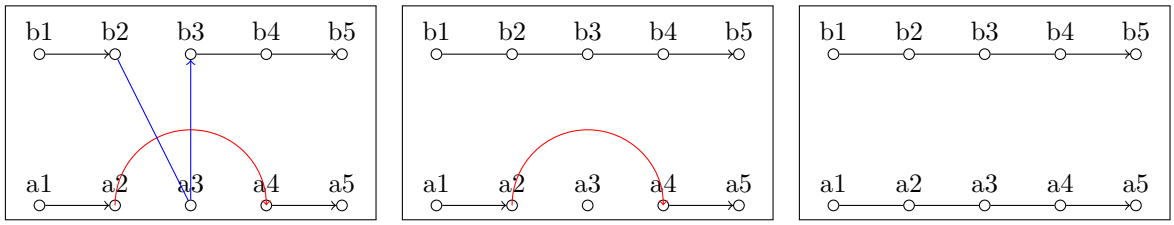

Figure 2.3: Illustration of the remove and add order

diversification. For the next moves, each time a move will be about adding an order back to a problem, those two variants will be considered.

On the figure 2.3, we can see an illustration where a3 is first removed from the route of b and then added to the route of a.

Here the first phase is considered as determinant for the tabu. As the moved order has supposedly been added back to its best place, it is not the move or the placing itself that is tabu but the order. The tabu value thus is simply the order ID of the removed order. It is forbidden to remove an order twice in a row as it would lead to an infinite loop.

3: Remove and add an order among the n worst orders

This method is basically the same as the previous one but, instead of selecting systematically the worst order, we select one order at random among the n worst orders that have been extracted. This addition of some randomness could help escaping some local minima we could obtain otherwise but could also keep us from finding a good solution as shown in the section 1.3.3.

Again, here, keeping only the order ID of the removed order in the tabu is enough.

The complexity is the same as the previous method since the only change is about the randomization : O(n).

4: Ejection chain

In the previous point, removing an order in a route will leave a virtually empty spot. We can try to place there another order from another route. We will call ejection chain here the fact that, for a given removed order, we will try to put another order at its previous spot. After having moved the other order, we will end the move by adding back the first order to the problem to make it consistent again.

In fact, this move adds another step to the two for the previous move.

1. Select one of the n orders to be removed from its route.
2. Select another order somewhere to take its place.
3. Try to add the original order on every other route, at every place.

```
For every route
  Check the gain of removing the order
  If the gain is the best so far, keep that order in mind
Remove the worst order from its route
For every order of other routes
  Check the gain of moving the order to this spot
If the move of one order is beneficial
  Do the move
For every route
  Select the best new placement for the order that was removed
```

On the figure 2.4, we can see an illustration of the ejection chain algorithm.

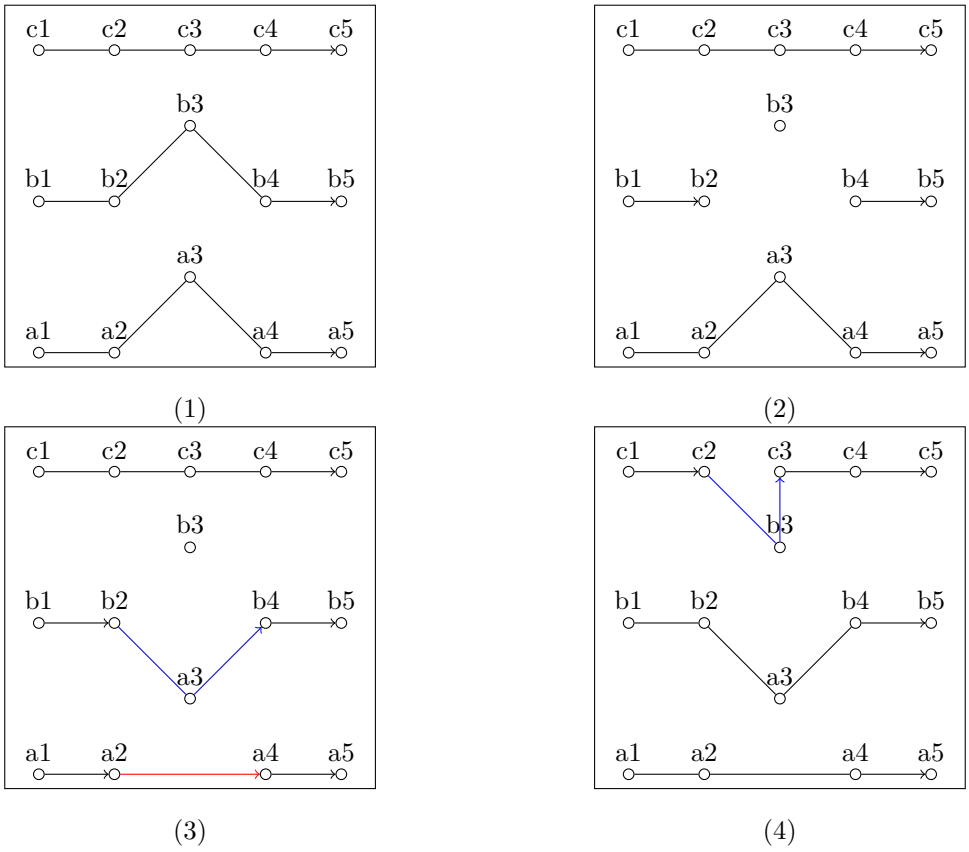

Figure 2.4: Illustration of the ejection chain

- Step 1 : The order b3 is removed from its route.
- Step 2 : The order a3 is added to the route b in between b2 and b4
- Stem 3 : The order b3 is added back to the problem on route c

The complexity is O(n) because all the operations are separated. O(n) for the removing (step 1), O(n) for the addition of the another order (step 2) and then O(n) for the last addition to the solution.

The value kept in the tabu list is only the ID of the order that was removed at first.

5: Longer chain

In the previous move, placing an order in the available spot leaves another spot empty. Instead of just moving a single order before placing the main one back to the problem, we can also look if doing the operation of placing another order in the newly available spot could be a good idea. After that, try to fill the new empty spot and so on... before finally adding the main order back to a route.

The length of the chain may vary and can be fixed or considered to be infinite until no improvement is possible any-more. Of course, we need to remember which orders have been locally moved during this step to prevent it from returning to its original spot and creating a possibly infinite loop. Another list is thus needed to prevent from doing unnecessary moves.

The only thing that changes for the complexity is the number of operations done for the second step. Thus there is a greater factor to O(n) but not higher degree of complexity.

Again, the value kept in the tabu list is only the ID of the order that was removed at first.

6: Several removals and addition at random (several random)

This method is based on the same principle as the *removing and add an order* method (section 2.4.2) but instead of just removing one only order, we remove n of them (for our experiments, n will be set to 3). Once removed, we add them back the the route where they fit the best in a random order. So basically, this move allows more flexibility by removing several "bad" orders at the same time. This also could add more diversity to the problem and escape local minima.

The complexity is only O(n) since we only need to go trough the n orders and keep track of the worst ones before adding them again. (O(n) + i*O(n))

For this move, the ID's of all the removed orders are added to the tabu list (respecting of course the tabu capacity).

7: Swap subroutes

We have proposed a search move consisting of swapping two orders from two different routes. Another possibility would be to swap two complete portions of the routes and see the impact on the objective function.

The sub-routes don't necessarily need to be of same length but they have to dispose of at least one order (otherwise this becomes the search method would become the sub-route moving exposed in section 2.4.2).

Figure 2.5: Illustration of swapping two subroutes

On the figure 2.5, we illustrate an example where the sub-routes (a3) and (b3, b4) are swapped from their respective routes.

The tabu value will be (as for the first search method) the starting order of both sub-routes. They will be kept in the the tabu list.

The complexity is higher than for most of the other methods. Indeed, as for the first method, the sub-routes are related to each other and the two phases are not performed just one after another.

We have to check n orders and n placings, which already gives us a complexity of $O(n^2)$. But when we have to check an order, we also have to check every possible length of sub-route, which multiplies by n for the first route. To swap with this sub-route, we also have to check every length of sub-route. This gives us an absolute maximum of $O(n^4)$. Hopefully, this values is never reached in practice since there is no possible way for all routes to be of size n. Furthermore, there are some criteria that reduce considerably the mean length of the sub-routes such as the fact that the orders have different sizes of bag. When trying to add a big order to a small capacity route, the sub-route would stop.

8: Remove and add a sub-route

Similarly, we can perform the search to move sub-route from one vehicle to another one. It is thus the same as the second search method but taking not only one order but a suite of orders. This method could lead to better results but can also lead to subsequently way bigger computing time. Indeed, we don't know how far we would need to go in the sub-route length.

This method is also really similar to the previous one at the exception that a the sub-route is not swapped with another but moved into another route. Again, there is a lot of operations to do to compute this method.

Here, the thing that is kept into the tabu is the ID of the order starting the sub-route.

Figure 2.6: Illustration of the remove and add subroute

An illustration can be found on the figure 2.6 where the subroute (b3,b4) has been removed and placed on the route a.

The complexity of this method will be the same as the previous one, except that there is no second sub-route to check. The algorithm will test n orders with their possible sub-routes of length of maximum n and try to place them at n other locations. The complexity will thus be $O(n^3)$.

An alternative to that would be to select the spot in every route where the waiting time is the biggest and to try to add sub-routes at this spot. The results would be the same.

2.4.3 Ejection chain, another approach

We have already explored an ejection chain move. But we also have test another way of implementing the idea of that search. This approach will follow the guidelines of the paper proposed by Gendreau and al. [5]. This paper shows analysis of an ejection chain approach for a local search on dynamic vehicle dispatching problem with pick-up and deliveries (reminder: our problem is stacker crane problem). The main difference with the problem that we have here is that the packages were considered really small and thus that the capacity of the vehicles was not a issue. On the contrary, our vehicles dispose of a unitary capacity. It adds a big constraint but also simplifies the fact that we don't have to deal with pick-ups and deliveries separately.

Comparison to the previous procedure

The phases of the ejection procedure are not exactly the same as the ones involved in the ejection chain developed before. Instead of ejecting the most costly order and then replacing it with another, the algorithm works as follow :

1. Select two requests i and j
2. Eject the request j from its route
3. Move the request i to the route of j

The difference is subtle but instead of considering the ejection and the insertion separately, their selection is done in pair. This is thus the best couple that is selected and not the best spot and then the best replacement.

As the approach is different, the complexity also differs. The number of possible ejection move is $O(n^2)$, besides, if we add the cost of the evaluation function of the route at each step, the complexity becomes $O(n^3)$.

On the contrary of what is explored in the paper, there is a lesser need of an approximation function because the computation of the cost of the insertion doesn't need to explore the possibilities of adding the pick-ups and deliveries of one order between the ones of another order.

Associated substructure

This other approach uses a separate structure which we will define here. There are two possible operations that are taking parts in the new ejection chain : ejection of an order by another one and insertion of an order into a route without ejection. To now select which of them to perform during the local search, we will use an associated structure. This structure is an another graph with the following elements :

- k layers, each of them representing one route
- n vertices corresponding each to one order
- k dummy vertices corresponding each to one route

Each layer contains every order that is assigned to it in the current solution plus the dummy vertex corresponding to their route.

The operations correspond to the connections between the vertices of those layers :

1. Ejections arcs (i,j) : each ejection arc corresponds to the ejection of the order j from its route and its replacing by the order i. The cost associated to this kind of arc is

   $cost$(cost of the insertion of i to the route of j when j has been removed)$-gain$(removal of j from its route). To compute those arcs, the program then checks every pair of order and computes its associated cost.

2. Insertion arcs (i, dummy) : those arcs correspond to the simple insertion of an order to another route without an ejection. They are linked to the dummy vertices to symbolize their addition into the route. Their cost is simply

   $cost$(cost of the insertion of i to the route of the dummy vertex) - gain(removing i from its current route).

3. Intra layer arcs : those arcs just connect each order to their dummy vertex. This potentially allows to execute several chain of operations at one step of the local search. Their cost is 0 because they correspond to the order staying on the route it already is on.

Those arcs are obviously directed as the ejection of j by i does not necessarily gives the same gain as the ejection of j by i. A visualisation of a part of this structure is shown on figure 2.7. On this figure, there are two layers (a and b) and we have the dummy edges in black, the ejection edges of a1 to the b layer in blue and the insertion of a1 into b in green.

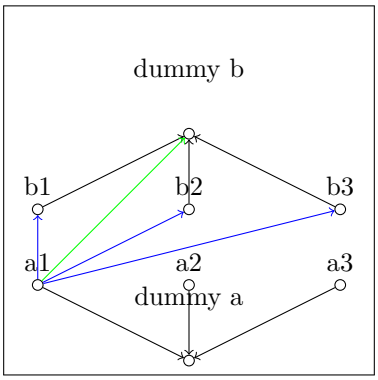

Figure 2.7: Associated structure for the ejection chain

To save ourselves from work, bugs and testing, we used an existing code to handle the associated graph part (under the license GNU General Public License, version 3 (GPLv3)) ([13] and [12]).

Finding the best moves

From this structure, the graph can be explored to find the moves that will be operated on the routes. A path as good as possible will be searched to select a suite of ejections or insertion moves. This path is constrained in the way that it cannot go several times trough the same layer. Indeed, the costs computation would be completely wrong in that case since orders wouldn't be at the same places as when the cost was computed. The cost of the edges wouldn't be accurate at all any more.

A simpler version of the path finding as been tested. The algorithm will find the edge associated with the best gain and follow the edges bringing also gain and select the best path out of it.

The tabu value has been set to be only the destination of an arc, which can be the route ID for an insertion or the order to be ejected for an ejection.

2.4.4 Complexities

Search method	1	2	3	4	5	6	7	8
Complexity	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n^4)$	$O(n^3)$

Table 2.1: Summary table of the complexities

A summary table of the time complexities are found in the table 2.1. Of course, to those basic complexities, we have to add the cost of computing the objective value of a route when we use it in the algorithm, which is the number of orders on that route ($O(n)$ in the worst case, but this a route with the n orders is far from likely).

2.4.5 Meta-heuristics

As stated before, meta-heuristics have been used to try to escape local minima.

Tabu search

For the different local search moved exposed earlier, some elements had to be selected to represent the moves and prevent them to be executed again for a certain amount of time. Those elements have been explained in the previous section along with their respecting move.

The application of the meta-heuristic is quite simple : when considering the moves in the neighbourhood, a tabu move is simply ignored.

For the basic tabu strategy, fixed values of the tabu list have been tried and compared. The results can be found in the third chapter of this paper.

Reactive tabu search

A reactive tabu approach has been tested. As explained in the preamble, the strategy is to not fix the size of the tabu structure but to adapt it regarding the current quality of the solutions. When a solution has been already explored, the tabu size is increased so that more moves are forbidden. This allows the search to explore other further search spaces.

To check for already explored solution, a new structure has been created. When a state is explored, a unique identification string is generated and its hashcode is kept in the structure.

As suggested in the paper [6], we need lower and upper bound for the size of the tabu structure to ensure that at least one move is possible. The bounds have been set to :

$$\begin{cases} \text{lower bound} = min(2, \text{number of moveable orders}) \\ \text{upper bound} = 0.85 * \text{number of moveable orders} \end{cases}$$

The upper bound set as number 0.85% is directly consequence of the experimentations we have done. As we will see in the results chapter. It allows new solutions to be found while approaching a good value for some searches.

The update of the tabu size is thus simply done by a simple verification :

```
Reactive tabu:
Convert solution to hashcode
If hashcode is already know
  tabuSize = min(max(tabuSize + 2, tabuSize * 1.2), maxTabuSize)
Else
  tabuSize = max(min(tabuSize - 2, tabuSize * 0.8), minTabuSize)
  put the solution hashcode to the know solutions
```

Simulated annealing

As explained in the previous chapter, simulated annealing consists in accepting or rejecting moving to another solution based on a simple probability approach. The approach proposed by the paper [6] has been largely applied. The acceptance probability of the solution is :

$$P_{\tau_i}(s'|s) = \begin{cases} 1 & \text{if } \Delta \geq 0 \\ e^{(\Delta/\tau_i)} & \text{otherwise} \end{cases}$$

And we have set the parameters as follow :

$$\begin{cases} \Delta &= f(s) - f(s') \\ \tau_0 &= 0.4 \\ \tau_{i+1} &= \alpha\tau_i \text{ with } \alpha = 0.95 \end{cases}$$

τ_i is updated every number of iteration proportional to the number of moveable orders.

```
Simulated annealing:
Keep previous solution in memory
Do a local search loop
If new solution is better (Delta >= 0)
  Accepts it
  Sets tau to original value
Else
  Applies the probabilities for acceptance
  If not accepted
    Restore the previous solution
  If at a right number of iterations
    Updates tau
```

This search combined with the tabu search prevents prevents the annealing from repeating the same moves over and over since. Indeed, even if the new solution is not accepted, the moves to reach it are put in the tabu list.

2.4.6 Adaptive memory

To try to enhance the quality of our solutions, we have tested an approach consisting of restarting and restoring solutions from a pool of solution. This approach will thus escape from a local minima trying to take another path in the search space by partly restarting the search.

The main idea is to keep a pool of solution when doing the local search. A fixed number of solutions states can be kept in a repository of routes : the adaptive memory. Every fixed number of iterations, the local state of the search is dropped and restored from those in-memory solutions. Routes from the solutions are selected at random to be restored. The algorithm is designed in a way that better solutions with better objective value would have higher chances to be reconstituted. Obviously, as the routes in the different solutions are also different, there might be overlapping between them. If a route overlaps one that has already been restored, it is dropped

from the restoration. At the end, all the orders that were not restored (which routes occurred in an overlapping) are added back to the problem at the best place using the insertion heuristic as it it was their first addition to the problem.

To reconstitute a solution from the ones in the memory, the algorithm works in several phases :

```
For every route
  Checks if one (or more) element of this route has already been restored
  If no
    Restore the route completely
  If yes
    Do not restore the route
Add all the orders that are missing from the solution
```

As the adaptive memory length is limited, the algorithm cannot afford to keep solutions that are the same, or even too similar. For that reason, a solution will be kept in the memory only if it is considered "different enough" from the one already kept (their objective values are different enough). When a new solution is to be added and the maximum length is reached, the worst solution is simply dropped.

After restoring the solution, the best solution found so far will be kept in memory but the pool of solutions for the current local search will be cleared to start the search only from that current solution.

A special attention has to be done when working on the dynamic version of the problem. When dropping the routes, some of them will certainly contain orders that were already or being processed. They cannot be added anywhere in the problem. For that reason, when a route is dropped, its fixed orders are still restored as well. As explained in the section 2.3.2, when fixing an order, the algorithm fixes it from the best solution in memory. Since none of our local search methods is allowed to touch to blocked orders, we are sure that they will stay at the same position as in that solution.

2.4.7 Impose a minimal number of orders

This concern was risen by the company. In practice, we can hope that it won't really change the output of the algorithm a lot.

If only lateness was playing a role in the objective function, there would be a very low probability that a delivery man would have to perform a significantly lower number than others. However, there is a huge difference in distance if a vehicle simply stays at the depot while other ones continue their deliveries each next to the next ones. The objective taking that into account, one delivery man can be forced to stay at the depot.

To impose a minimal number of numbers for the vehicles to perform, we have slightly modified our algorithms to promote the addition of an order to the routes that the least number of orders. We try to add first orders to the routes with a smaller number of orders. If no compatible route is found, we try the addition on the routes with one more order and so on. This ensures that a certain equality is achieved among the order repartition.

In the next algorithm, we can see that if every vehicle doesn't perform 3 orders yet, we apply the method proposed above. This won't absolutely impose that the minimum number of deliveries is reached but it will guarantee that, if it is possible, it will be done.

```
Addition back with a minimum number of orders:
min = minimal number of orders a any route
while an insertion is not found
  For every route
    If the size of the order and the route are compatible
      If the number of routes on this route equals to min
        Try the insertion on this route at every place
        If the insertion is possible and the gain is better than the previous one
          An insertion is found
          Keep as best insertion if necessary
  If no insertion was found
    min = min +1
Do the best insertion
```

Chapter 3

Presentation of the results

3.1 Results

We will now present the results from implementing and testing our different algorithms aiming to optimize the objective of the VRP.

3.1.1 Static problem

We have done first some experiments on the static problem to have an idea of the efficiency of some key aspects of the algorithm such as the initialization and the tabu search behaviour.

Initial solution

As previously stated in the paper, time has not really been spent on finding a good initial solution. However, a significant boost in initial solution quality has been found by trying to find an initial solution in a less greedy way. The first solutions only adds orders at the end of the routes where the delivery man is available the earliest and the second one scans the routes to add the order at the best possible place.

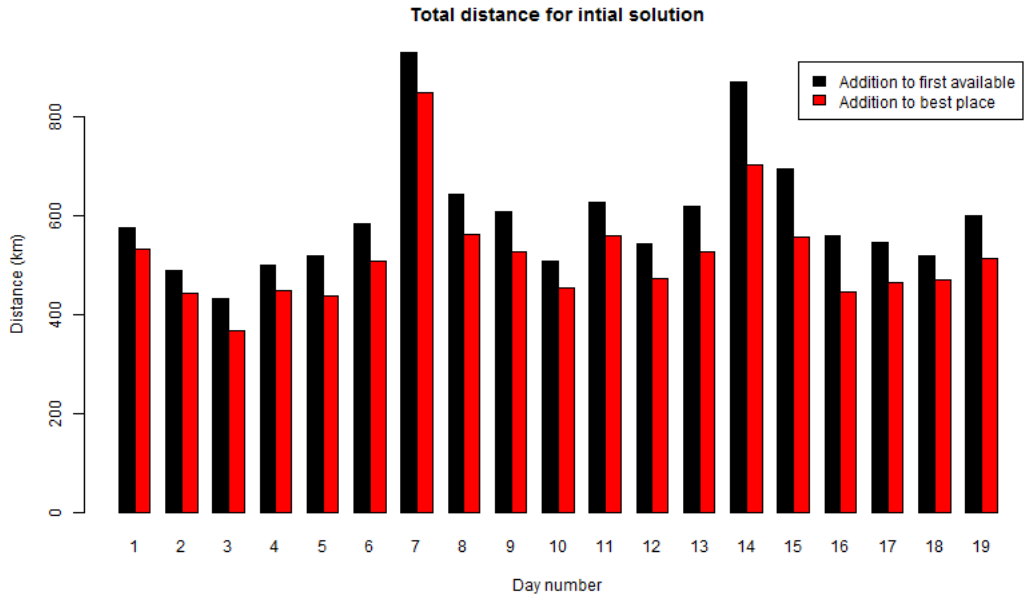

Figure 3.1: Summary of distances for initial static solutions for the two methods

It is obvious on the figures 3.1 and 3.2 that the second method leads to globally better results. This is thus this strategy that will be used for any insertion based move in every further result. The graphs for the objective value and overtime can be found in appendices.

We do not bring graphs or evidence to show it here but the results for applying local search on the first initial solution were barely as good as the second initial solution itself. This simple improvement has came late in the analysis and made us recompute a lot of results but the outputs were proven worth it.

Tabu length analysis

As stated before, the local minima are a big issue regarding the local search. The first step was thus to look for the ideal size of the tabu list for each resolution method. In this section, each graph will show the objective value evolution (mean for all days) per iteration for each local search

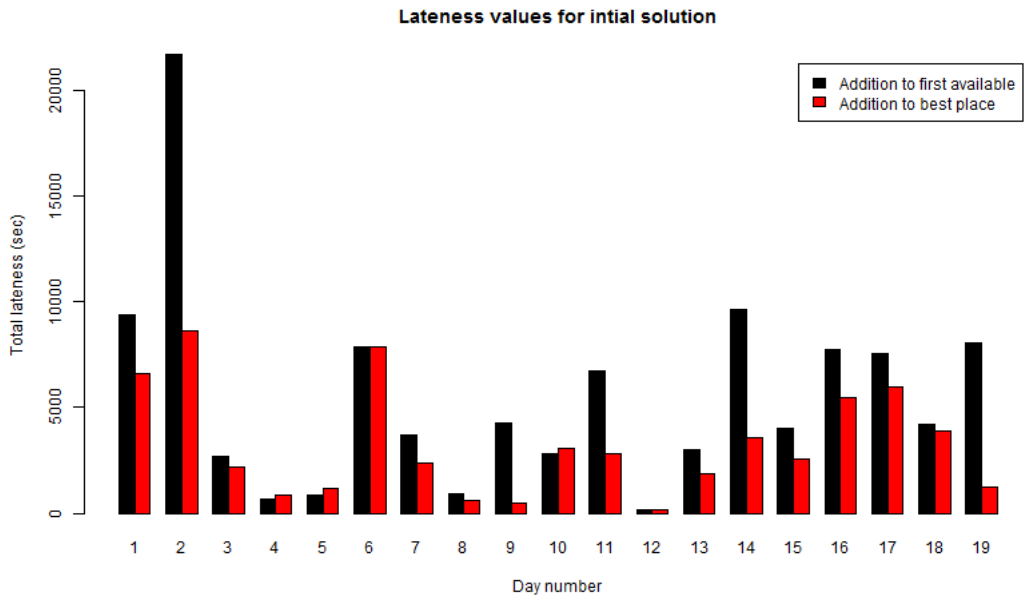

Figure 3.2: Summary of latenesses for initial static solutions for the two methods

method proposed earlier, regarding to the size of the tabu list. Each tabu value corresponds to a percentage of the total number of orders that have to be processed on a given day. The tabu length will be equal to this percentage * length of moveable orders. Graphs showing this evolution for the first day only can be found in appendix.

1. **Simple swap**

   As said in the previous section, the time complexity of the simple swap method is quite high. This method is thus quite heavy in computation time. The tests here have thus been done using only 50 iterations. The graph shown on figure 3.3 shows the evolution of the mean of the objective value for all days that were obtained on the static problem, testing several values as the maximum size of the tabu list.

   Even tough the objective values are not that different, we can clearly see that small values are the best for this method. For the rest of the paper, we will thus keep the tabu length as 10% of the number of orders to be processed in the local search for that search method.

   This small value could be explained by the fact that tabu moves are involving several orders at once. Then swapping an order several times with other ones could be still improving the solution. If the order is in the tabu, we keep it from taking part in the next move and it leads to worse solution.

2. **Removing and adding one order**

   The figure 3.4 shows the results for the second method. As we can see, the best values are between 60 and 70% without that much variation. We will be choosing a value around 85% for this method.

3. Removing and adding one order chosen randomly between the best ones

   The figure 3.5 shows the results for the third method. Similar conclusions can be done with this one. Again, fixing the tabu length to 80% seems a good choice.

4. **Ejection chain with max 1 element**

   Again, the graph on figure 3.6 leads us to choosing a value around 75%.

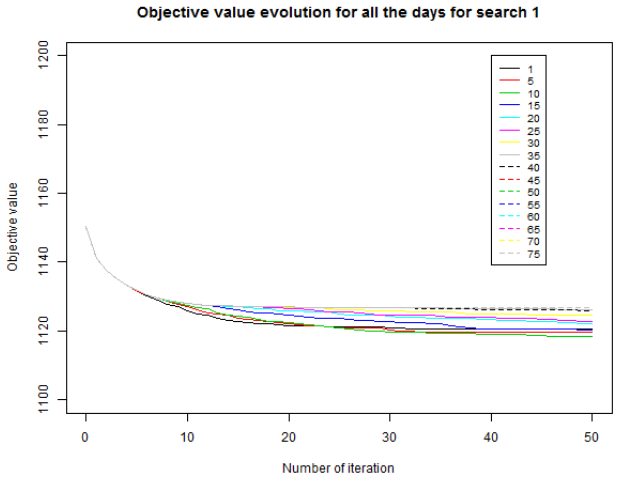

Figure 3.3: Evolution of the mean the objective value for all days for several tabu percentages for first search method

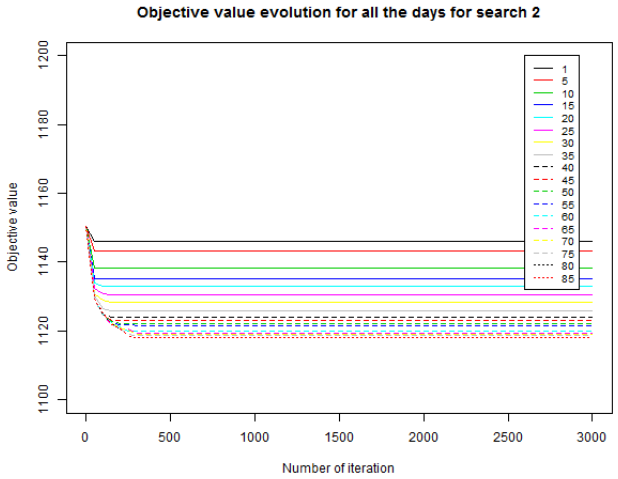

Figure 3.4: Evolution of the mean the objective value for all days for several tabu percentages for the search method 2

5. **Ejection chain of max 3 elements** The tabu value here will be between 70 and 85% as well (figure 3.7). The final value will be arbitrary set to 80%.

6. **Removing several orders at once**

   The graph (3.6) leads us to a value around 65%.

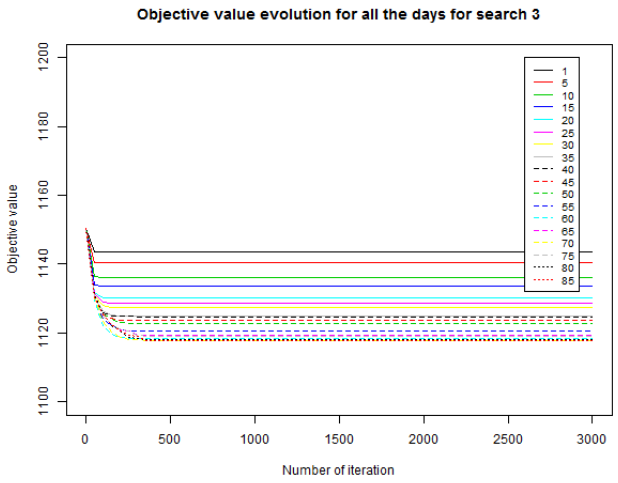

Figure 3.5: Evolution of the mean the objective value for all days for several tabu percentages for the second method 3

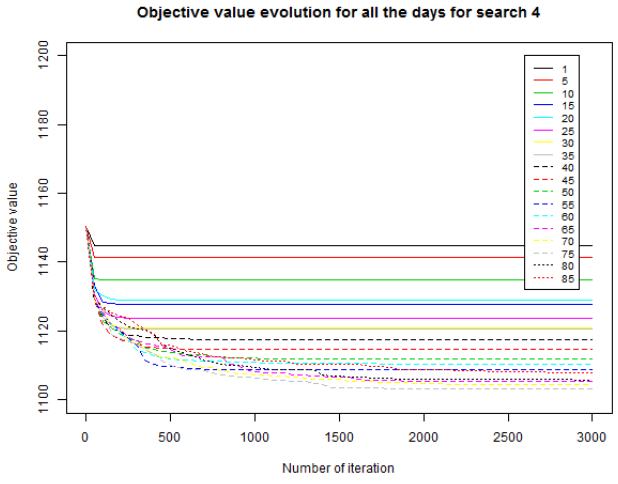

Figure 3.6: Evolution of the mean the objective value for all days for several tabu percentages for the search method 4

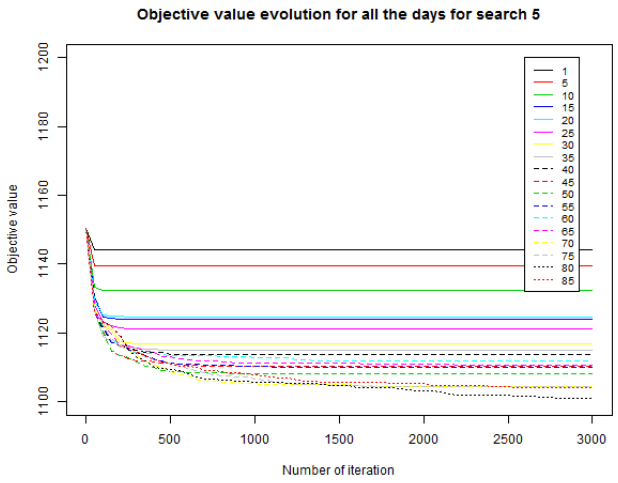

Figure 3.7: Evolution of the mean the objective value for all days for several tabu percentages for the search method 5

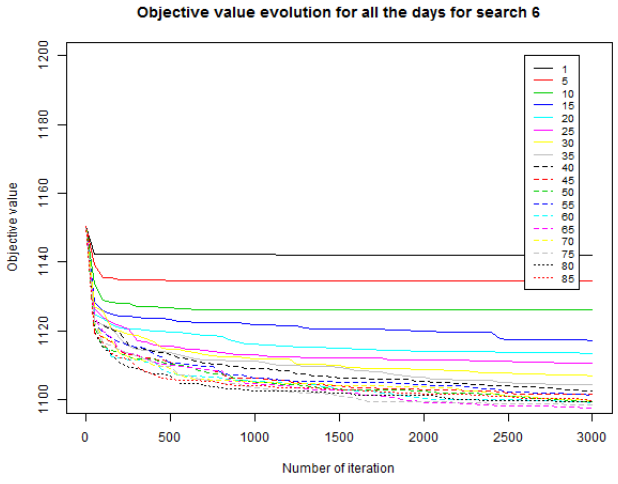

Figure 3.8: Evolution of the mean the objective value for all days for several tabu percentages for the search method 6

7. **Subroute swap**

   The method 7 method was the sub-route swap. Similarly to the simple swap, it is way more heavier in complexity than most of the other ones. The number of iterations was thus not set to 3000 but only 50. As for the first search method, big tabu sizes lead to worst results. For this reason, the tabu percentage will be only set to 5% to try to get the best possible moves.

   The same hypothesis can be done as for the first search to explain this behaviour.

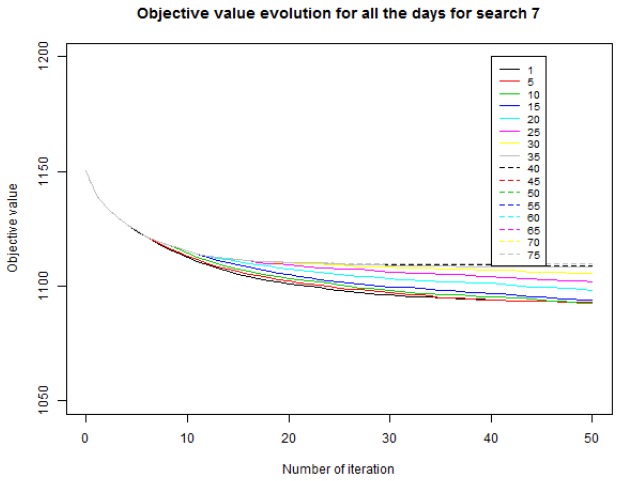

Figure 3.9: Evolution of the mean the objective value for all days for several tabu percentages for the search method 7

8. **Subroute remove and add**

   This method was also only performed for 50 iterations. As we can see on the figure 3.10, the tabu value doesn't really seem to affect the search much. However, a small value seems to be preferred.

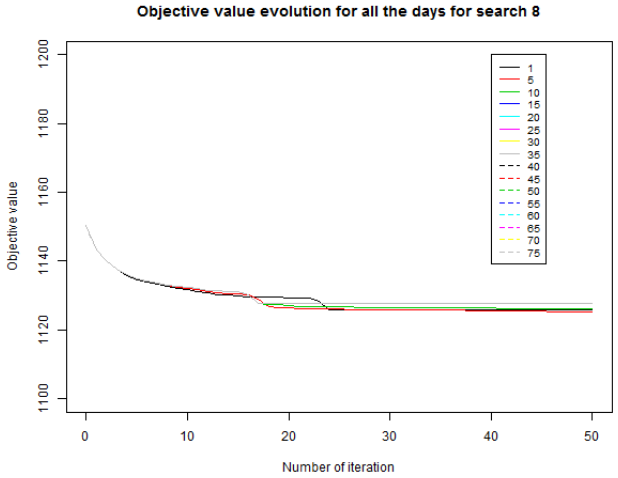

Figure 3.10: Evolution of the mean the objective value for all days for several tabu percentages for the search method 8

9. **Ejection chain with associated graph**

   The last method to be tested was the variation of the ejection chain move. It is also a slow method. The graph on figure 3.11 shows that a tabu value of around 10% is to be preferred.

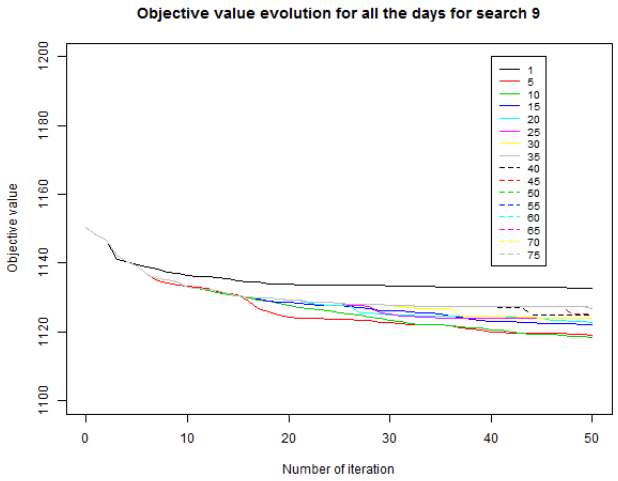

Figure 3.11: Evolution of the mean the objective value for all days for several tabu percentages for the search method 9

10. **Variant of the searches 2 to 6 with the forbidding to add an order back to its previous route in the addition**

    The figure 3.12 shows the tabu analysis for the second search method (removing and adding one order) but with the restriction of not being able to add an order back to the route from which it was previously removed. The results are different than for the the first bersion and a value around 20% seems to be preferred.

    All the other searches (3 to 6) with that restrictions have shown similar results. The graphs can be found in the appendix (figure 5.12 to 5.15).

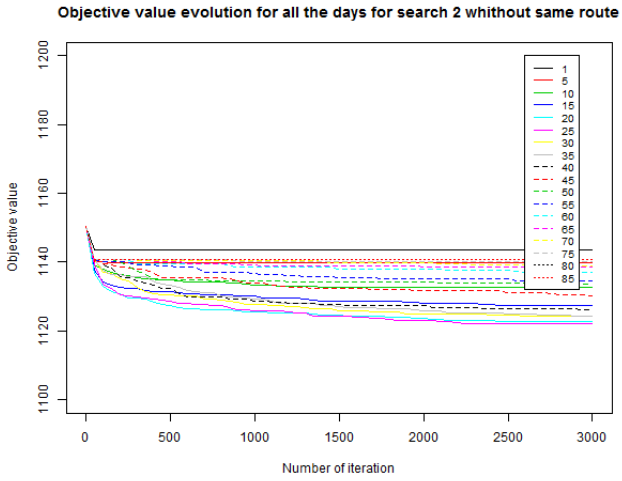

Figure 3.12: Evolution of the mean the objective value for all days for several tabu percentages for the search method 2 without possibility of adding the order to its previous route

A summary table of the tabu percentages can be found in table in table 3.1. Those values will be used for the further results when the reactive tabu meta-heuristic is not activated.

Search	1	2	3	4	5	6	7	8	9
Tabu	10	85	80	75	85	65	5	5	10
Tabu when no addition back		20	20	20	20	20			

Table 3.1: Summary table of the tabu values

Performance comparison

One thing could already be observed during the tabu analysis : some methods seems to give better results than others. But as expected, their time complexities are very different and some methods also run really slower. Using the tabu values found earlier, this section will thus compare the evolution of the objective value not with a fixed number of iterations but with a fixed running time. This approach is more egalitarian regarding to our objective. After all, we are interested in methods that will run in real time, their computing speed is thus one the most important factors.

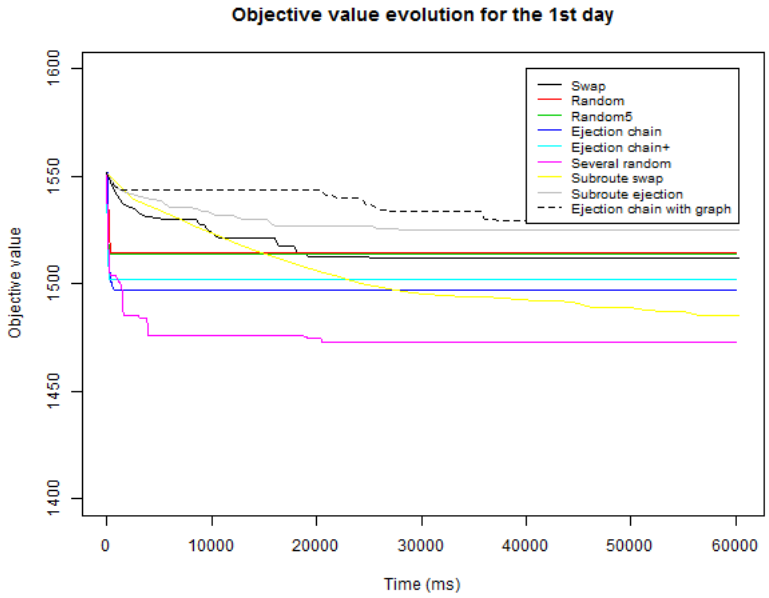

Figure 3.13: Evolution of the objective value for the first day for a 60s execution

The figure 3.13 shows the evolution of the objective value for the different search methods for an equal execution time of 60 seconds on the static problem. 60 seconds seems realistic for a computing time on one state of the problem if we were dealing with the dynamic version of the problem. Besides, we can see that most of the methods already seemed to stagnate. The graph only represents the evolution for one day because, as the time between each iteration was not exactly the same for each search, doing a mean and putting it on one single graph was not as easy. However, the graphs for the other days are available in the appendix (figure 5.16 to 5.23).

We can see on this graph that there are one method that seems to take the lead : the ejection of several orders followed by their reinsertion in the problem. Besides that, the ejection chain seems to also give intermediate results. We can see that the sub-route swap starts slow but finally overtakes the ejection chain results. We might want to put some of the methods away for the dynamic problem but the results might differ a lot when having smaller search spaces.

If we look at the graphs of other days as well, it is clear that "Several random" ejection globally gives better results. However, the more we let the local search run, the more the sub-route swap go closer to its best solution and then sometimes tend to surpass it.

We can conclude for now that if we need a very quick and good solution, the "several random" ejection is the better solution but if we can afford more execution time, a sub-route swap could be better.

We also have tested the variations of the search methods 2 to 6. On the figure 3.14, we have grouped the mean final results on the static instances for all the untouched methods (on the left) and the modified methods (on the right). We have added a reference line to make the comparison

easier. We can see that the variants of the search are globally giving worse results than their normal variant.

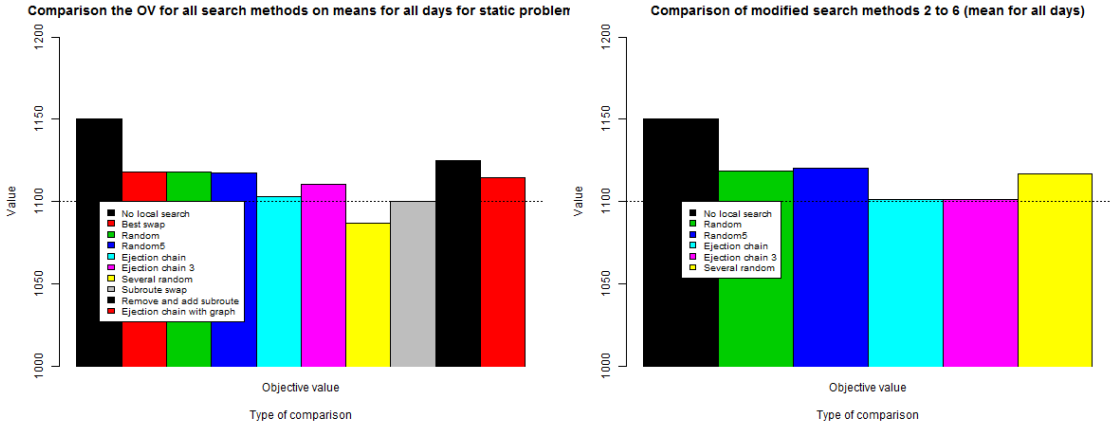

Figure 3.14: Final objective value for the mean of all the days for a 60s execution

Application of meta-heuristics

During the experiments, to have a significant boost, we had to apply the reactive tabu and simulated annealing simultaneously. We will thus consider here that "applying the meta-heuristics" means using both reactive tabu and simulated annealing together. The figure 3.15 shows the objective value evolution for an instance of the problem. We can clearly see that the objective evolution doesn't stay in local minimums and seems to dispose of a good diversification.

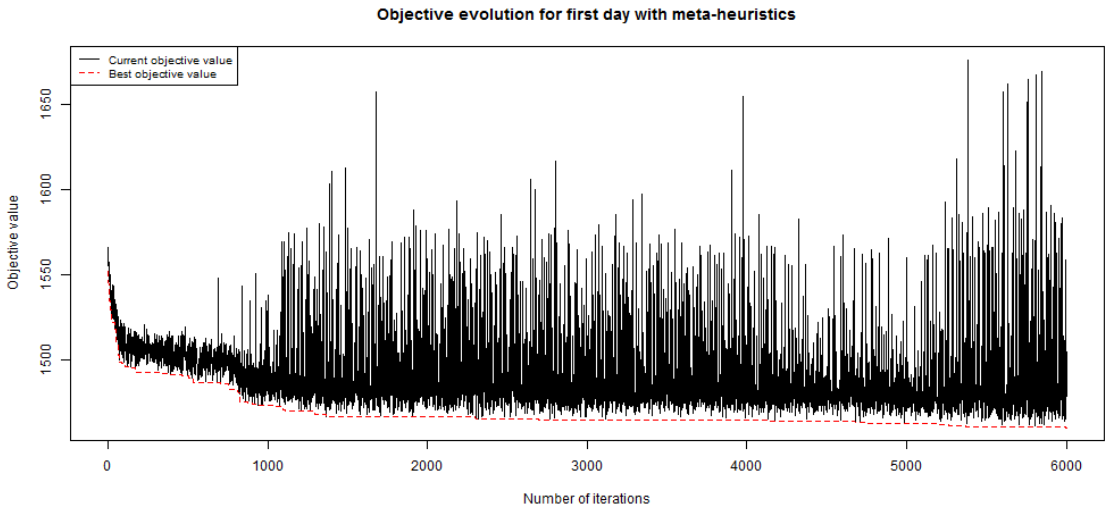

Figure 3.15: Objective value evolution for day one using meta heuristic

After multiple tests, it has been found that the application of the meta-heuristics on the variant of the search methods 2 to 6 gave good results, in fact similar to the ones found by the several random ejection without meta-heuristic. Indeed, we can see on the figure 3.16 that the methods are now all under the symbolic reference line. With such promising results, all we can do is hope that the impact on the dynamic solution will be as good.

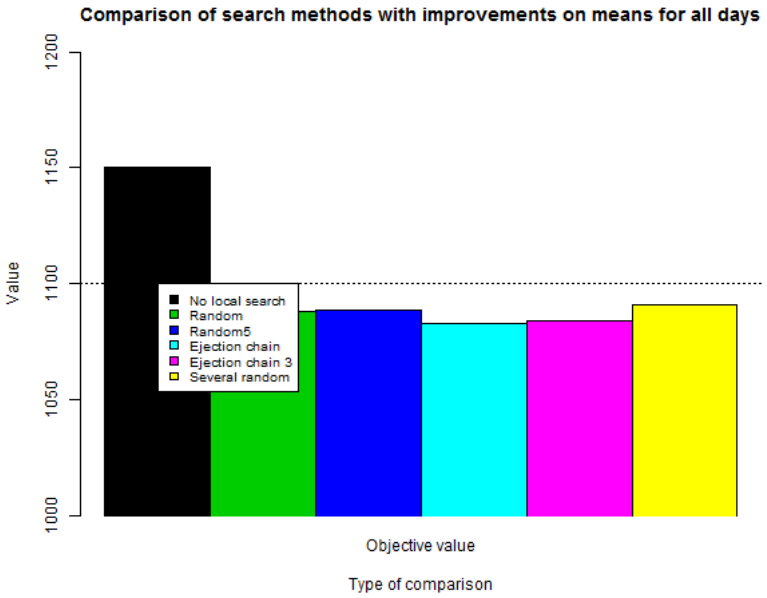

Figure 3.16: Final objective value for the mean of all the days for a 60s execution

Application of adaptive memory

Unfortunately, the results applying the adaptive memory were always either very similar significantly worse than the ones got without applying this strategy. This strategy will thus be put aside. We won't show results here since they are not showing improvements.

3.1.2 Dynamic problem

As mentioned numerous times in this paper, the dynamic problem offers a different perspective. Indeed, all the orders are not moveable at the same time. So most of the time, the number of orders that the local search has to deal with is really low. A short analysis on how many orders are generally processed simultaneously is done in section 3.1.3. As such, the results of the searches might vary much. We have run our prototype on every day of the data allowing the searches to run a time proportional to the time between two events (order or delivery). The proportional factor is the same for every search routine and thus leads to a roughly equal amount of computing time for each of them.

3.1.3 Conclusion from the data

The figures 3.17 illustrates what we said about the number of orders being moveable at the same time for a typical day. The black curve shows the evolution of orders passed on this day and the red curve represents the orders that are assigned an thus blocked. The gap between them represents the orders that are moveable at a given time. We can see that they are never above 50% of the total number of orders and sometimes is near 0. This clarifies the fact that the search space is thus really smaller for the dynamic version, which justifies the fact the the dynamic solution could hardly be as good as the static one.

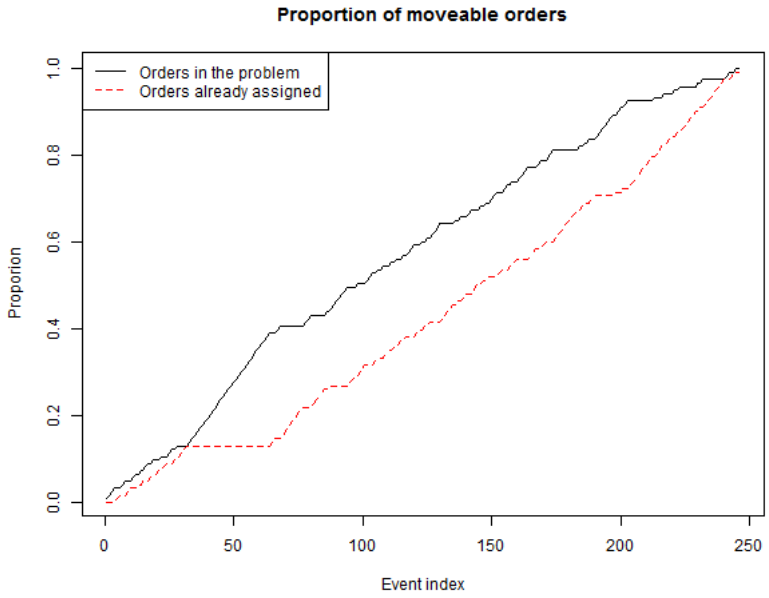

Figure 3.17: Illustration of the evolution of proportion of number of moveable orders for one day

3.1.4 Results without meta-heuristics

The figures 3.18 to 3.21 show the values of the objective function and the different parameters that are taken into account at the end of the computation of the dynamic problem by the prototype.

The conclusions are without appeal. There are two groups of methods : the ones based on swap operations and the ones based on ejection. On the dynamic version of the problem, the methods based on ejections outputs far better results. Since they have smaller time complexities, it is not a surprise that they can achieve better results with de dynamic problem.

A result that is also interesting to take into consideration is the local worst lateness on each route. The graph 3.22 shows us that this value doesn't vary much regardless of the search method

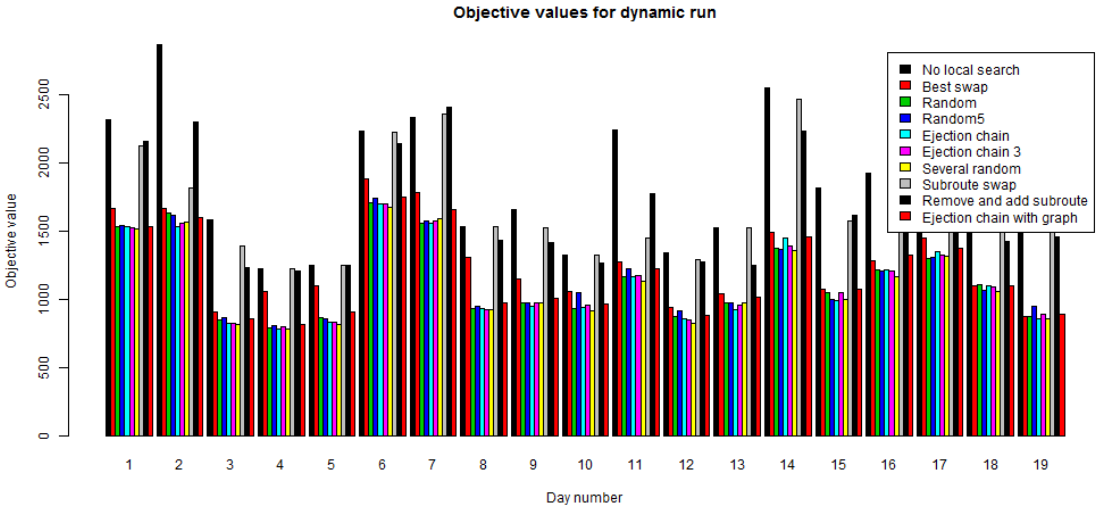

Figure 3.18: Comparison of the final objective value after a dynamic run

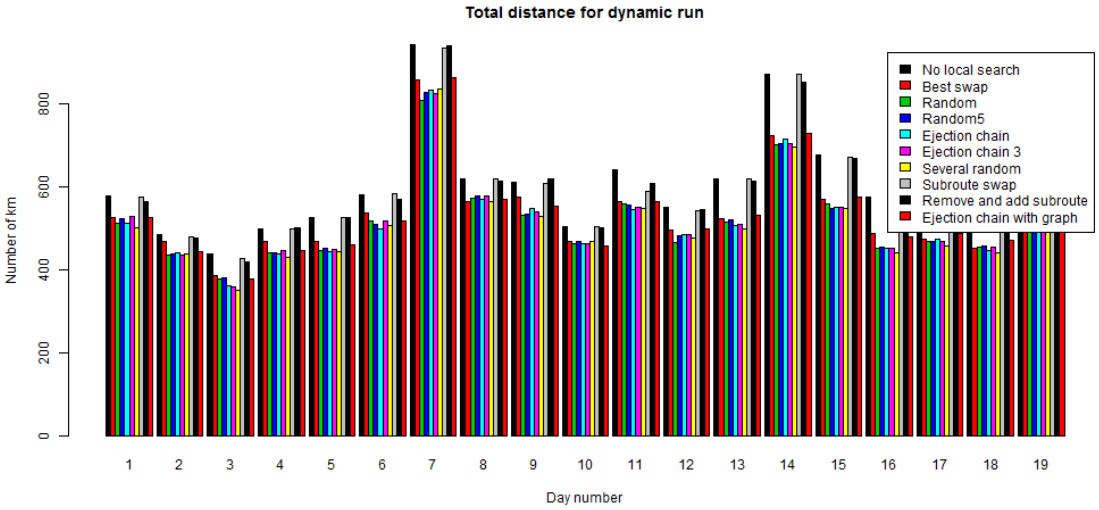

Figure 3.19: Comparison of the final distance after a dynamic run

considered. This is most probably caused by the fact that too many orders are performed at a similar time and thus that an actual solution with better worst lateness management would be hardly impossible.

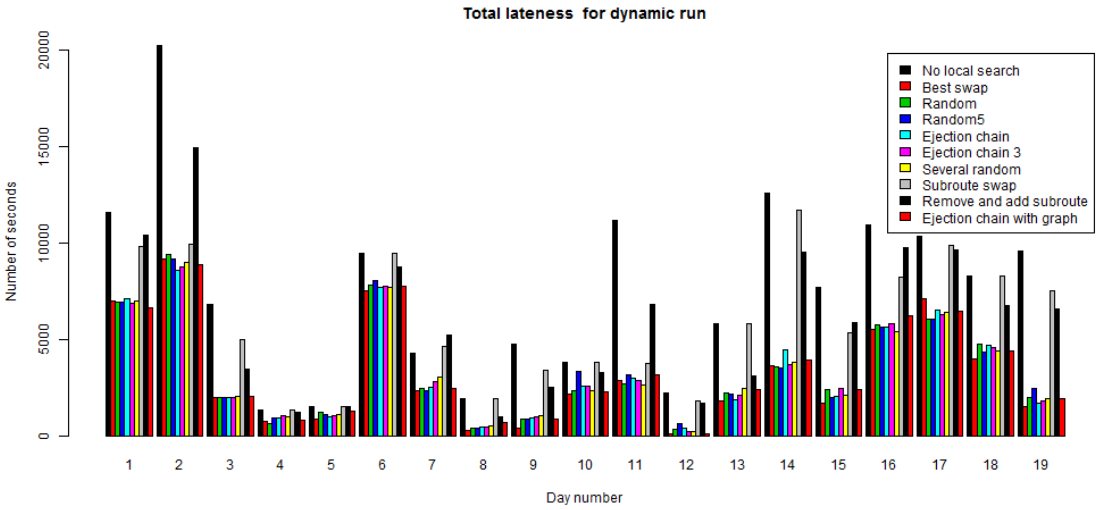

Figure 3.20: Comparison of the final lateness after a dynamic run

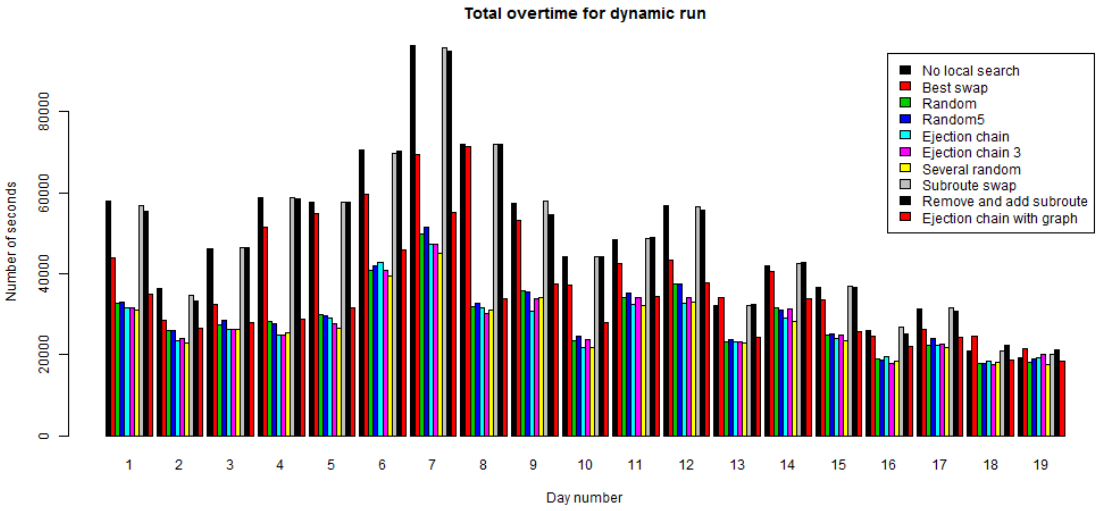

Figure 3.21: Comparison of the final overtime after a dynamic run

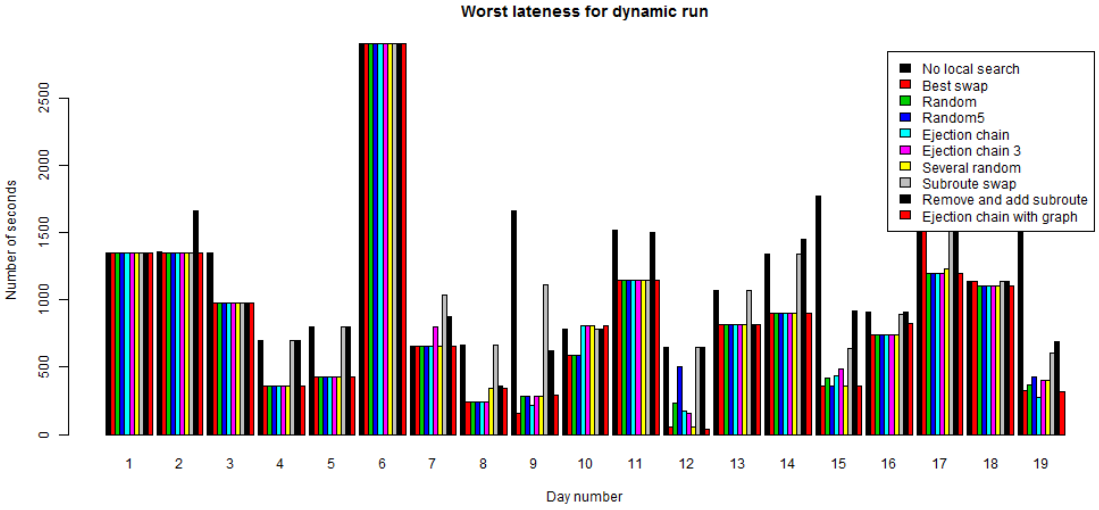

Figure 3.22: Comparison of the final worst lateness after 60s runtime

The figure 3.23 shows a summary of the results when considering the mean of each value. As the overtime doesn't really play an important role, we have put it away. On the contrary, we have shown the worst lateness because it is a big concern even tough it is not in the objective.

On this figure, the same conclusions can be done. However, the simple swap method slightly outperforms the other ones regarding the lateness but is behind for the objective value.

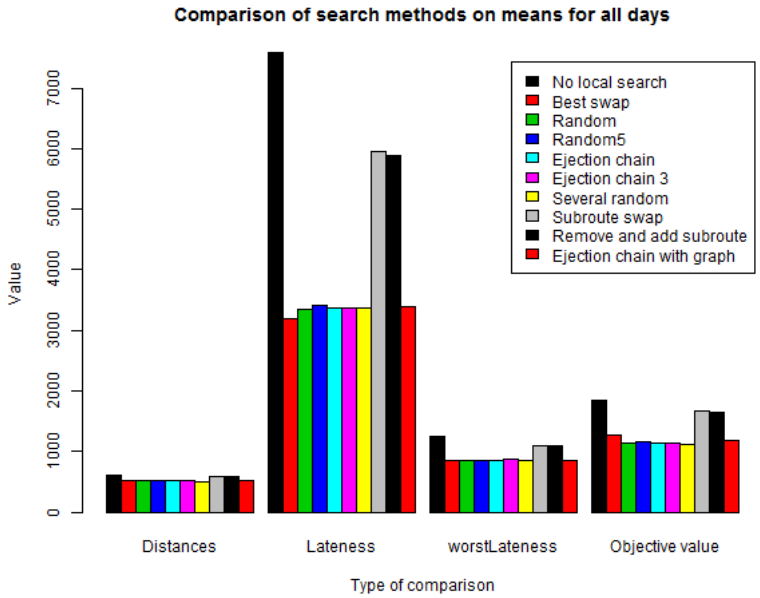

Figure 3.23: Comparison of the values for the mean on all days

The table with the results can be found in figure 3.2. For each parameter, the value is compared with the objective value got when not applying local search at all ("search 0"). A lesser percentage means better results since we want to minimize the objective value.

The geometric mean is then done to compare them objectively. It is clear, now with numbers, that the algorithms based on ejection work better on the dynamic problem. For now, the best one being the several random ejection.

Search	0	1	2	3	4	5	6	7	8	9
Objective Value	1839.98 1.00	1268.94 0.69	1141.89 0.62	1156.65 0.63	1130.38 0.61	1136.27 0.62	1118.42 0.61	1666.73 0.91	1655.84 0.90	1178.83 0.64
Distance	601.03 1.00	532.02 0.89	514.05 0.86	518.61 0.86	514.73 0.86	516.46 0.86	508.56 0.85	593.13 0.99	592.31 0.99	529.40 0.88
Lateness	7597.26 1.00	3195.58 0.42	3356.74 0.44	3418.58 0.45	3367.89 0.44	3374.47 0.44	3368.11 0.44	5946.00 0.78	5881.47 0.77	3390.89 0.45
Overtime	47922.47 1.00	41736.68 0.87	29217.00 0.61	29617.74 0.62	27885.84 0.58	28235.47 0.59	27304.89 0.57	47900.53 1.00	47538.74 0.99	31033.84 0.65
Worst lateness	1257.58 1.00	851.53 0.68	845.37 0.67	859.42 0.68	846.58 0.67	865.47 0.69	853.11 0.68	1101.79 0.88	1098.89 0.87	851.37 0.68
Geometric mean	1.000	0.687	0.626	0.634	0.617	0.625	0.616	0.908	0.900	0.645

Table 3.2: Summary and comparison of the the results for dynamic problem

On the figure 3.24, we can see that the modified versions of the search methods 2 to 6 give very similar results. To make better comparison between all the results, we show the table for the results with the modified algorithms in the table 3.3. By comparing the values in the two tables 3.2 (regular search methods) and 3.3 (search methods 2 to 6 modified), we see that once again, the modified versions of the searches give slightly lower results that the ones without the modification.

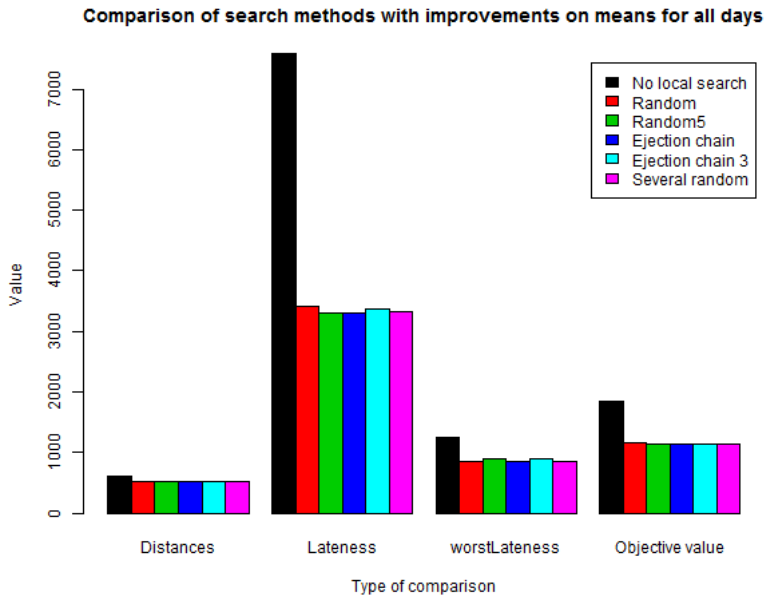

Figure 3.24: Comparison of the values for the mean on all days

Search	0	2	3	4	5	6
Objective Value	1839.98 1.00	1154.02 0.63	1128.91 0.61	1133.89 0.62	1141.51 0.62	1129.08 0.61
Distance	601.03 1.00	517.80 0.86	511.70 0.85	514.11 0.86	514.93 0.86	513.30 0.85
Lateness	7597.26 1.00	3265.21 0.43	3383.58 0.45	3355.58 0.44	3273.32 0.43	3296.68 0.43
Overtime	47922.47 1.00	29378.32 0.61	28643.16 0.60	28937.00 0.60	28905.32 0.60	28360.11 0.59
Worst lateness	1257.58 1.00	862.26 0.69	886.84 0.71	844.63 0.67	887.00 0.71	857.47 0.68
Geometric mean	1.000	0.628	0.630	0.624	0.628	0.617

Table 3.3: Summary and comparison of the the results for dynamic problem with modified searches 2 to 6

With meta-heuristics

The use of the reactive tabu and simulated annealing have provided a huge boost in the results for some methods but not all. Typically, the modified versions of the methods based on order ejection (search methods 2 to 6, with the prevention of putting back on order on its previous route directly) have shown similar but great results.

Search	0	2	3	4	5	6
Objective Value	1839.98	1119.85	1134.37	1120.89	1112.08	1112.15
	1.00	0.61	0.62	0.61	0.60	0.60
Distance	601.03	509.40	508.97	509.60	508.87	507.49
	1.00	0.85	0.85	0.85	0.85	0.84
Lateness	7597.26	3265.21	3383.58	3355.58	3273.32	3296.68
	1.00	0.43	0.45	0.44	0.43	0.43
Overtime	47922.47	28392.74	28704.16	27572.42	27588.42	27499.42
	1.00	0.59	0.60	0.58	0.58	0.57
Worst lateness	1257.58	857.84	846.21	851.37	849.63	844.68
	1.00	0.68	0.67	0.68	0.68	0.67
Geometric mean	1.000	0.617	0.625	0.618	0.613	0.608

Table 3.4: Summary and comparison of the the results for dynamic problem using meta-heuristics

We can see on the table 3.4 that the results are indeed better than before, the best one going down to 60,8% of the value without local search. However, the improvement doesn't seem that impressive since it's compared to the values obtained when applying no local search at all. The next table shows the improvement by comparing the best ones got "raw" with the best ones got with the meta-heuristics (table 3.5).

Search	Objective value	Distance	Lateness	Overtime	Worst lateness
Best dynamic without meta-heuristics	1112.1531	507.4905	3296.6842	27499.4211	844.6842
Best dynamic with meta-heuristics	1118.4165	508.5570	3368.1053	27304.8947	853.1053
Improvement (%)	0.5631779	0.2101517	2.1664526	-0.7073836	0.9969525

Table 3.5: Comparison of the the results for dynamic problem using and not using meta-heuristics

The improvement is very small but visible. As it is done on the mean of all days, it can be considered as significant. Besides, the comparison is done on similar methods but not exactly same ones ince the addition strategy is not the same for the two best methods for the two cases.

3.1.5 Adaptive memory

As for the static version, the application of the adaptive memory algorithm was not convincing. They thus won't be shown here.

3.1.6 Imposing a quota of orders

Some methods have shown big disparity regarding orders distribution to the delivery man. Unfortunately, maintaining such a constraint comes with a cost. In fact, the methods maintaining the most disparity are the methods applying ejection moves, the same methods that gave really good results just above.

To impose that a certain the number of orders is processed by every delivery man, we have tested the modified version of the insertion algorithm with those most promising methods. (see section 2.4.7)

Experiments have shown that the use of the meta heuristics are essential in finding a good solution in that case. Indeed, the results obtained without the application of the meta-heuristics were poor (very poor). They almost were the same as not applying a local search at all.

The best method once again was found to be the several random ejection. The next results will thus be the ones we got with this method, which were the best we were able to get. The other methods based on ejection were a little behind. The methods based on swaps were still really far behind too.

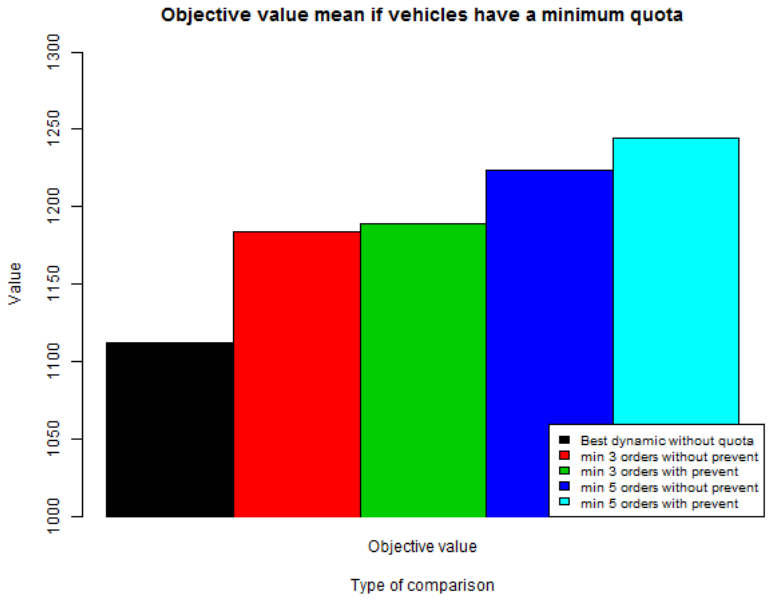

Figure 3.25: Comparison of the final objective value when imposing a quota

The figure 3.25 shows the objectives values that were achieved with the two versions of the "several random" algorithm (with and without preventing to add back on order to its previous route) while imposing a better equality in the order distribution. For those results, we have tried to maintain a minimum of 3 and then 5 orders per delivery man. The results are obviously worse when imposing this additional constraint. However, the impact on the objective value is far less that we could think of. It seems then realist to accept it as a good solution.

This time, the basic version of the several random (allowing to add an order to its previous route when removed) is the one giving better results.

More details on the distances, lateness and other parameters can be found in the next section with the figure 3.26 showing the different values compared to the company solution.

3.2 Comparison to official solution

The figure 3.26 shows a comparison of the values we got from reconstituting the assignments of the orders as computed by the company and the results we obtained with our prototype, both imposing a quota of order and not.

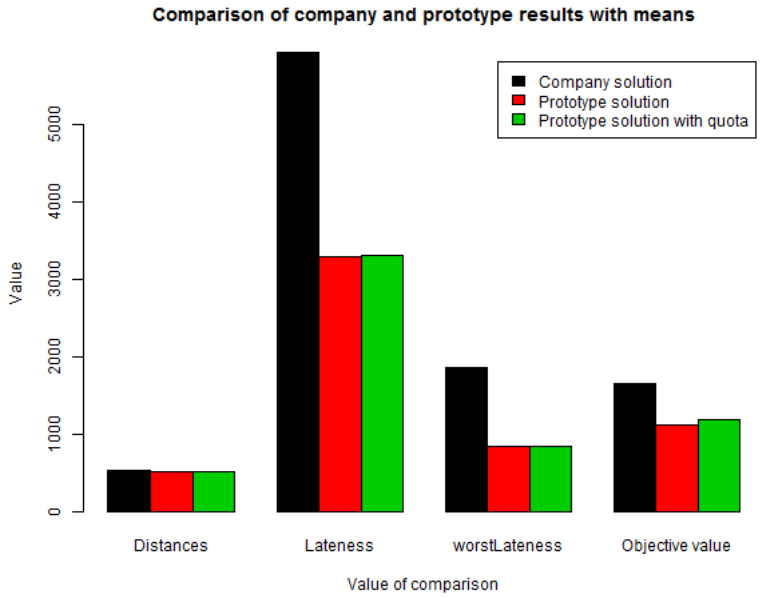

Figure 3.26: Comparison of the different elements for the company and prototype solutions

To have a view of the values for every day, see the figures 5.25 to 5.27 in the appendix. The graphs speak for themselves : our solution is better than the one from the company. Applying the constraint of having at least 3 deliveries per delivery man slightly decreases its quality but the resulting output is still very satisfying.

However, let's not forget that the company solution was obtained dealing with the real world and not a simulation. The differences can thus results from external real world issues such as traffic jams (less probable if the deliveries are done by bike) or restaurants and customers being not as punctual as in an ideal world. The conclusions here look very promising but are to be observed with caution.

3.3 More improvements

Some other concerns have been raised :

- Would it be good to allow multi-orders bags to the vehicles ?
- What would change if the main concern was the worst lateness on the deliveries for one day and not the global lateness ?

3.3.1 Multi-orders delivery

Allowing to deliver several order at once completely changes the definition of the problem. As seen in the section 1.2, if we respect the formal definitions in the literature, the problem would switch from a stacker crane problem to a vehicle routing problem with pick-up and delivery. Some approaches might still work but as we have seen in this paper, the methods suggested for the the VRPPD are not necessary good for the SCP. Thus, the opposite might as well be true (and most probably will). This means that, to respect that constraint, some big changes may be necessary.

Even if some of the work (the prototype for example) could doubtlessly be partly reused, some huge adaptations would have to be done such as a modification of the objective function computation because the deliveries would be no longer forced to be performed right after their pick-ups. For the same reason, approximation functions would have to be used since computing the placing of the orders on the road wouldn't have only n possibilities any-more. For each order in the road, we would have to check the possibilities of interleaved pick-up and deliveries.

3.3.2 Concern about the worst lateness

The worst lateness already plays a big role in the objective function as it is. Besides, a huge worst lateness is most probably caused by a number of available vehicles too small to handle a big number of simultaneous orders or by the customer making an unrealistic request of ordering a meal from the other side of the city and asking it to be delivered in less than 15min.

Inserting the worst lateness in the objective and giving an even bigger importance will most probably lead to a deterioration of the other deliveries. Trying to reduce the worst lateness, its value would be spilled on the other deliveries. It is up to the company to decide if they want to have a systematic big lateness for every order of just have an exceptional significant lateness (maybe then exceptionally offering the delivery).

Chapter 4

Conclusion

4.1 Conclusion

4.1.1 Context

In this master thesis, we have analysed a problem consisting in delivering meals ordered from customers by fetching them to restaurants and then delivering meals at customers houses. This problem entered in the classification of *stacker crane problem*. The objective was to find a fast and efficient algorithm for this real-time real-world problem.

We have build a prototype capable of reproducing the situation of one day of activity of the company both for the static problem and the dynamic problem. This prototype applied local search procedures to improve the solution as much as we can.

4.1.2 First analysis

First analysis on the possible operations to be done have shown that there were actually two classifications of moves that could be done in the local search : moves based on swap and moves based on ejection and replacing. The first ones have shown to be too slow and not efficient enough for our real time problem.

We have explored two versions of the algorithms based on ejections : one version allowing to remove an order and putting it back on the same route and the second one preventing to do so. At first sight, the first version seemed to work better but further analysis could lead to the conclusion that it was not necessary the case. Applying meta-heuristics on all the algorithms have shown very motley results. Some promising search methods didn't really show great improvements by applying them. However, some that we excepted to be worse got a huge boost to even surpass the results of the promising methods. Besides, they were really important when imposing a minimum number of orders to assign to the delivery men.

Unfortunately, other explored methods such as the adaptive memory didn't seem to be significantly improving the solutions, even worse, they were sometimes worsening them.

4.1.3 Final results

Our final results are really promising since they are way better than the ones provided for reference by the company. To improve its productivity, the company should probably use a local search algorithm based on ejection (such as several random ejections) and addition back to the routes, being very careful of respecting the conditions of placing and blocking the orders.

The constraint of imposing a minimum number of deliveries for each delivery man didn't affect much the quality of the solution with the best algorithm found without this constraint.

If the application of those algorithms to the real world shows results as good as the ones presented here, everyone should be quite satisfied : the customers, the delivery men but also the company.

Bibliography

[1] Norbert Ascheuer, Sven O Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *STACS 2000*, pages 639–650. Springer, 2000.

[2] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

[3] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[4] Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

[5] Michel Gendreau, Francois Guertin, Jean-Yves Potvin, and René Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174, 2006.

[6] Kai Gutenschwager, Christian Niklaus, and Stefan Voß. Dispatching of an electric monorail system: Applying metaheuristics to an online pickup and delivery problem. *Transportation science*, 38(4):434–446, 2004.

[7] Florence Massen et al. *Optimization approaches for vehicle routing problems with Black Box Feasibility*. PhD thesis, UCL, 2013.

[8] Martijn Mes, Matthieu Van Der Heijden, and Aart Van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59–75, 2007.

[9] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[10] Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[11] Warren B Powell. An operational planning model for the dynamic vehicle allocation problem with uncertain demands. *Transportation Research Part B: Methodological*, 21(3):217–232, 1987.

[12] Kevin Wayne Robert Sedgewick. Adjmatrixedgeweighteddigraph.java. http://algs4.cs.princeton.edu/44sp/AdjMatrixEdgeWeightedDigraph.java.html, 2015. Accessed: 05-07-2015.

[13] Kevin Wayne Robert Sedgewick. Java algorithms and clients. http://algs4.cs.princeton.edu/code/, 2015. Accessed: 05-07-2015.

[14] Doris Sáez, Cristián E Cortés, and Alfredo Núñez. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. *Computers & Operations Research*, 35(11):3412–3438, 2008.

[15] Martin Savelsbergh and Marc Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[16] Darsono Tjokroamidjojo, Erhan Kutanoglu, and G Don Taylor. Quantifying the value of advance load information in truckload trucking. *Transportation Research Part E: Logistics and Transportation Review*, 42(4):340–357, 2006.

[17] Jian Yang, Patrick Jaillet, and Hani Mahmassani. On-line algorithms for truck fleet assignment and scheduling under real-time information. *Transportation Research Record: Journal of the Transportation Research Board*, (1667):107–113, 1999.

[18] Jian Yang, Patrick Jaillet, and Hani Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2):135–148, 2004.

List of Figures

1.1 Graph corresponding to an instance of the problem 6
1.2 Illustration of the saving heuristic [7] 12
1.3 Illustration of the insertion heuristic [7] 12
1.4 Illustration of the sweep heuristic [7] 13
1.5 Illustration of local optimum [7] 14

2.1 Illustration of an event queue 27
2.2 Illustration of the swap move 31
2.3 Illustration of the remove and add order 32
2.4 Illustration of the ejection chain 33
2.5 Illustration of swapping two subroutes 34
2.6 Illustration of the remove and add subroute 35
2.7 Associated structure for the ejection chain 36

3.1 Summary of distances for initial static solutions for the two methods 41
3.2 Summary of latenesses for initial static solutions for the two methods 42
3.3 Evolution of the mean the objective value for all days for several tabu percentages for first search method 43
3.4 Evolution of the mean the objective value for all days for several tabu percentages for the search method 2 43
3.5 Evolution of the mean the objective value for all days for several tabu percentages for the second method 3 44
3.6 Evolution of the mean the objective value for all days for several tabu percentages for the search method 4 44
3.7 Evolution of the mean the objective value for all days for several tabu percentages for the search method 5 45
3.8 Evolution of the mean the objective value for all days for several tabu percentages for the search method 6 45
3.9 Evolution of the mean the objective value for all days for several tabu percentages for the search method 7 46
3.10 Evolution of the mean the objective value for all days for several tabu percentages for the search method 8 46
3.11 Evolution of the mean the objective value for all days for several tabu percentages for the search method 9 47
3.12 Evolution of the mean the objective value for all days for several tabu percentages for the search method 2 without possibility of adding the order to its previous route 48
3.13 Evolution of the objective value for the first day for a 60s execution 49
3.14 Final objective value for the mean of all the days for a 60s execution 50
3.15 Objective value evolution for day one using meta heuristic 50
3.16 Final objective value for the mean of all the days for a 60s execution 51
3.17 Illustration of the evolution of proportion of number of moveable orders for one day 52
3.18 Comparison of the final objective value after a dynamic run 53
3.19 Comparison of the final distance after a dynamic run 53

3.20 Comparison of the final lateness after a dynamic run 54
3.21 Comparison of the final overtime after a dynamic run 54
3.22 Comparison of the final worst lateness after 60s runtime 55
3.23 Comparison of the values for the mean on all days 56
3.24 Comparison of the values for the mean on all days 57
3.25 Comparison of the final objective value when imposing a quota 59
3.26 Comparison of the different elements for the company and prototype solutions . . 60

5.1 Summary of overtime for initial static solutions for the two methods i
5.2 Summary of worst latenesses for initial static solutions for the two methods ii
5.3 Summary of objective value for initial static solutions for the two methods ii
5.4 Evolution of the objective value for day one for several tabu percentages for search 1 iii
5.5 Evolution of the objective value for day one for several tabu percentages for search 2 iv
5.6 Evolution of the objective value for day one for several tabu percentages for search 3 iv
5.7 Evolution of the objective value for day one for several tabu percentages for search 4 v
5.8 Evolution of the objective value for day one for several tabu percentages for search 5 v
5.9 Evolution of the objective value for day one for several tabu percentages for search 6 vi
5.10 Evolution of the objective value for day one for several tabu percentages for search 7 vi
5.11 Evolution of the objective value for day one for several tabu percentages for search 8 vii
5.12 Evolution of the objective value for all days for several tabu percentages for search 3 modified viii
5.13 Evolution of the objective value for all days for several tabu percentages for search 4 modified viii
5.14 Evolution of the objective value for all days for several tabu percentages for search 5 modified ix
5.15 Evolution of the objective value for all days for several tabu percentages for search 6 modified ix
5.16 Evolution of the objective value for 2 days for the static search methods x
5.17 Evolution of the objective value for 2 days for the static search methods x
5.18 Evolution of the objective value for 2 days for the static search methods xi
5.19 Evolution of the objective value for 2 days for the static search methods xi
5.20 Evolution of the objective value for 2 days for the static search methods xi
5.21 Evolution of the objective value for 2 days for the static search methods xii
5.22 Evolution of the objective value for 2 days for the static search methods xii
5.23 Evolution of the objective value for 2 days for the static search methods xii
5.24 Evolution of the objective value for 2 days for the static search methods xiii
5.25 Comparison of the total distance of the company solution versus the dynamic results xiv
5.26 Comparison of the total lateness of the company solution versus the dynamic results xiv
5.27 Comparison of the worst lateness for an order of the company solution versus the dynamic results xv
5.28 Comparison of the objective value for an order of the company solution versus the dynamic results xv
5.29 Comparison of the overtime for an order of the company solution versus the dynamic results xvi

List of Tables

2.1 Summary table of the complexities 37

3.1 Summary table of the tabu values 48
3.2 Summary and comparison of the the results for dynamic problem 56
3.3 Summary and comparison of the the results for dynamic problem with modified searches 2 to 6 57
3.4 Summary and comparison of the the results for dynamic problem using meta-heuristics 58
3.5 Comparison of the the results for dynamic problem using and not using meta-heuristics 58