A constructive and improvement heuristic for the dial a ride problem

Auteur : Thomas, Frédérick
Promoteur(s) : Paquay, Célia
Faculté : HEC-Ecole de gestion de l'Université de Liège
Diplôme : Master en ingénieur de gestion, à finalité spécialisée en Supply Chain Management and Business Analytics
Année académique : 2021-2022
URI/URL : http://hdl.handle.net/2268.2/13737

HEC LIÈGE
Management School - Liège Université

A CONSTRUCTIVE AND IMPROVEMENT HEURISTIC FOR THE DIAL A RIDE PROBLEM

Jury:

Supervisor:
Célia PAQUAY

Reader:
Anisha MAHARANI

Master thesis by

Frédérick THOMAS

For a Master in business engineering with a specialized focus in Supply Chain Management and Business Analytics

Acknowledgements

First, I would like to thank Ms Célia Paquay my supervisor. She trusted me with his proposal of thesis and was always available to help me at all the stages of the process.

I would also like to thank the company Dufrais for allowing me to balance working for them and on my thesis. They allowed me to arrange my work schedule in order to have the time needed to finish my thesis.

Next, I want to thank Vincent Thomas and Orian Lambin for reading my thesis before it was summited. His remarks were very helpful.

Finally, I want to thank all my friend and family which support me during the making of my thesis.

Table of contents

1. List of abbreviations 6

2. Introduction 8

2.1. Problem description 8

2.2. Literature review 11

3. Methodology 16

3.1. Problem formulation 16

3.2. Algorithm description 20

3.2.1. Bee Algorithm 20

3.2.2. Deterministic Annealing 21

3.2.3. Hybrid Bee Algorithm with Deterministic Annealing 22

3.2.4. Decision-making 23

3.3. Algorithm implementation 24

3.3.1. Algorithm structure 24

3.3.2. Components 27

3.3.2.1. Instances 27

3.3.2.2. Solution structure 28

3.3.2.3. Feasibility check 29

3.3.2.4. Calculation cost 30

3.3.2.5. Solution creation 30

3.3.2.6. Selection phase 31

3.3.2.7. Improvement phase 32

3.3.2.8. Population adaptation 34

3.3.2.9. Stopping criterion and large operator 34

3.3.2.10. Parameters 35

3.3.2.11. Local operators 36

4. Results and improvements 44

4.1. Initial results 44

4.1.1. Performance of the algorithm 45

4.1.2. Cost distribution 46

4.1.3. Performance of the LNS 46

4.2. Improvements 47

4.2.1. Parameters tuning 47

4.2.2. Larger operator for Large Neighbourhood Search 51

4.2.3. Maximum ride time for vehicle 53

4.2.4. Time-dependent speed 54

4.3. Final version 56

5. Conclusions 58

6. List of Figures 60

7. List of Tables 62

8. List of Algorithms 64

9. Appendices _____ 66
10. List of Appendix _____ 78
11. Bibliography and References _____ 80

1. List of abbreviations

DARP: Dial a ride problem

BA: Bees Algorithm

DA: Deterministic Annealing

ACO: Ant colony optimisation

GA: Genetic algorithm

SA: Simulated Annealing

LNS: Large Neighbourhood Search

Avg: Average

Dev: Deviation

CPU: central processing unit

2. Introduction

The dial a ride problem (DARP) is a specific transportation problem. The objective is to optimise the planning of a collection of trips made by a fleet of vehicles. Those trips aim to satisfy requests from users while meeting various constraints.

DARP has received increasing attention since the end of the 1980s. This is due to the fact that they answer a growing need of the population. Indeed, public transportation has shown their limits and an intermediary solution between personal transport services and them is required.

However, from an optimisation point of view, DARP is hard to solve as they must handle a lot of constraints that can be different from a case to the next. That is why good algorithms are needed in order to guide companies and states in their logistics.

In this section, I will present the problem through his more famous example before going into his technical characteristics. Once this is done, we will take a look on the different algorithms used to solve it. This will be done thanks to a literature review focusing on the contributions of various papers concerning the method to solve DARP rather than on contributions regarding improvement of objective functions and constraints.

2.1. Problem description

The more common example of dial a ride problem is the transport of the elderly or disabled people. Due to their conditions, they might not be able to travel thought classic public transports, which are not equipped to handle their needs. Those needs can go from special accommodations to the necessity to have qualified drivers with medical knowledge. That is why several companies specialised themselves in the transport of people with specific needs. Moreover, several countries have developed dial-a-ride services.

For extreme cases, transport can be done with a single vehicle per person. However, dial-a-ride problems focus on vehicles that have the possibility to accommodate more than one person at the same time. That can be a minibus that has been arranged to accommodate wheelchairs or bed. Indeed, it is both cheaper and better for the environment to transport several people at the same time.

Due to the aging of the population, those kinds of services will be needed even more in the future. Furthermore, there is a trend for public transportation to fight global warming. It is then important to have specialised public transport.

Both of those problems are solved thanks to dial a ride transportation. Moreover, new ambulatory health care services are being developed every year.

All of those factors lead to an increase of the demand. Nowadays, the market has trouble absorbing it while maintaining reasonable costs. That is why good optimisation tools to solve DARP are indispensable. Otherwise, the price of those services will rise even though people needing those services already have a lot of costs related to their conditions.

As we can see through this example, one of the main differences between DARP and other classic transportation problems is the human factor. The aim is to minimise not only the cost of the transportation but also the inconveniences to the users. Those inconveniences include the desired hour to arrive at a specific medical appointment, but also the ride time. The well-being of the user is a key aspect of the service and an augmentation of cost can be justified if it increases the quality of the service. A balance between the two must be found.

On a more technical aspect, each user or request is represented by two nodes: an inbound and an outbound. The first correspond to the origin location where they want to be picked and the second one is the destination location where they want to be dropped off. Both have time windows during which they need to be processed in order to meet the demands of the user. Furthermore, users have a maximum ride time duration in order to maintain a certain quality of services.

The difficulty of the problem lies in how these requests can be assigned in the most efficient way. Each user must be allocated to a vehicle, and each vehicle must follow the best sequence in order to satisfy the different requests assigned to it. For that, a vehicle can transport more than one user at the same time, as long as it does not violate the maximum time duration of each user.

DARP can be represented by a graph of nodes which represent the pickup and deposit locations of each user. The journey of a vehicle is then represented by a series of arc linking those nodes together in a specific order. For example, in Fig 1, we have 6 requests with 6 corresponding pickups P and deliveries D. In the depot, we have two vehicles.

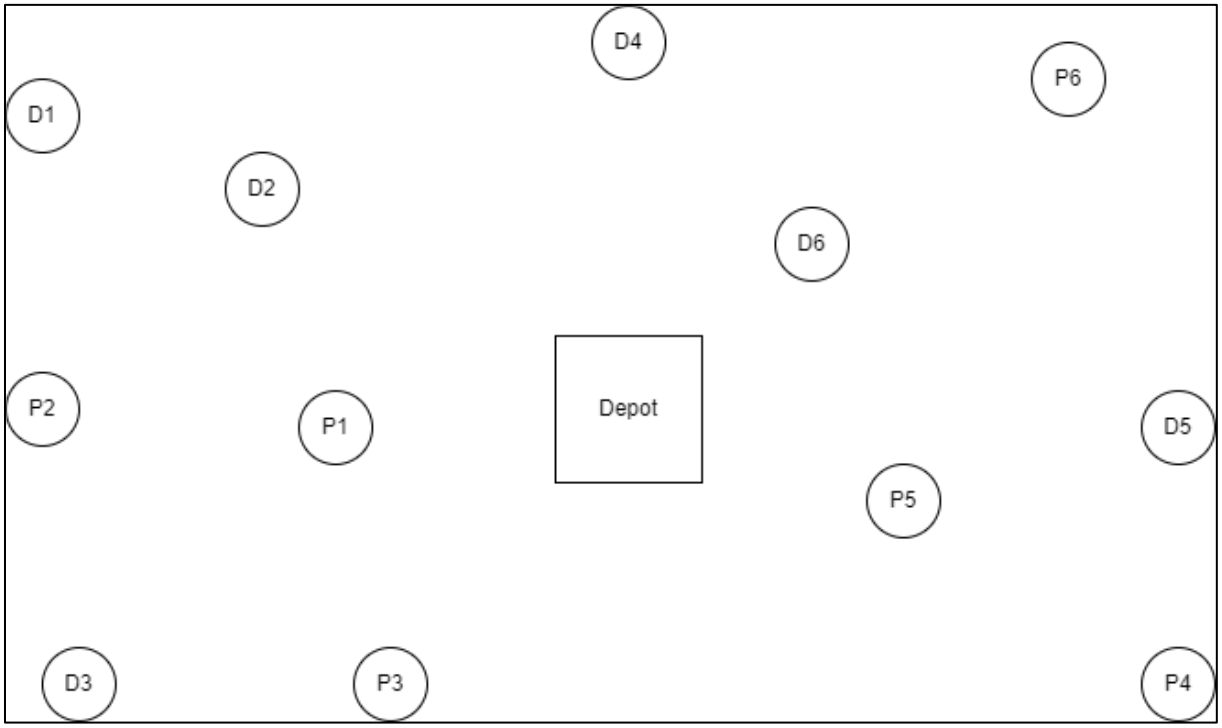

Fig 1. Empty example of DARP

A possible solution is presented in Fig 2 with the blue arrows representing the journey of the first vehicle and the black ones the journey of the second.

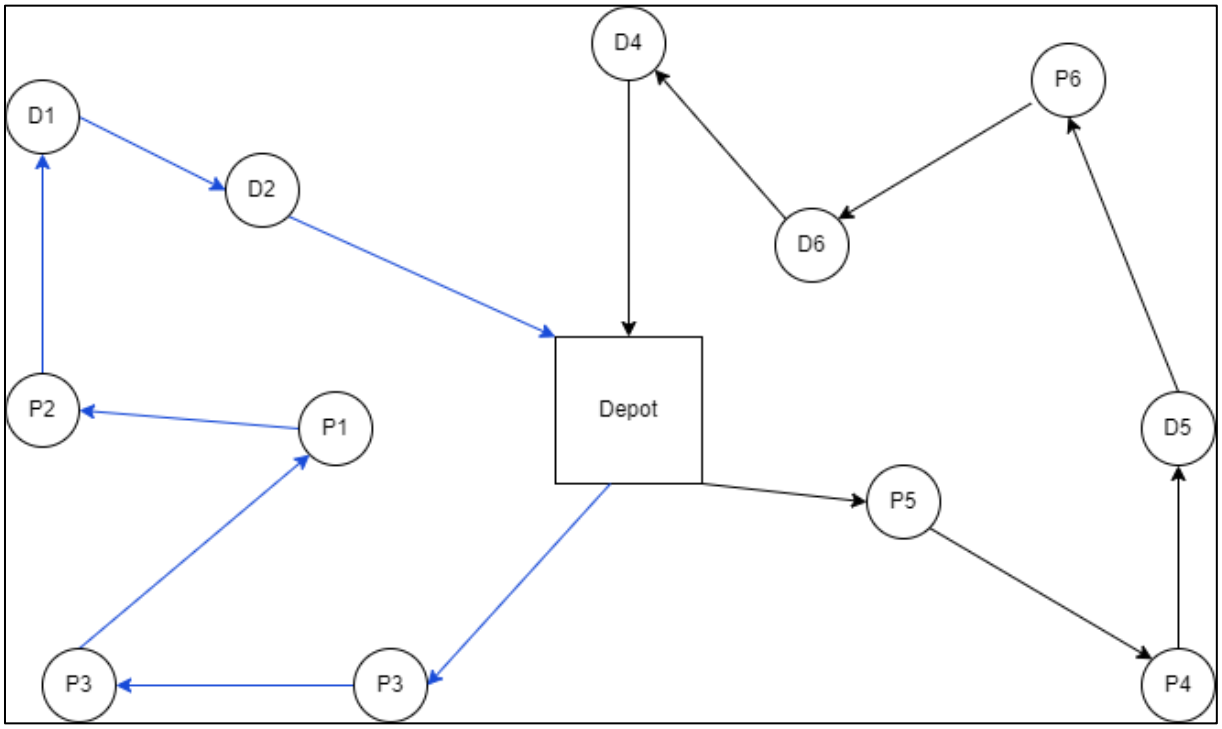

Fig 2. Completed example of DARP

The constraints and objective function can be different from paper to paper. This is due to the fact that the problem is complex. Furthermore, there is more than one aspect to this problem. The main purpose remains the minimisation of the cost, but several managerial aspects can be added such as the drivers' well-being or vehicle maintenance. Finally, the quality of service must be ensured by minimising the users' inconveniences.

The main variation of the objective function concerns the number of vehicles used. Some papers consider the number of vehicles to be only a constraint, while others take it as a variable to minimise. In practice the first guaranty the fulfilment of services while the latter follows more the point of view of a company that would need to control the cost of additional vehicles.

The constraints can be very different depending on the resemblance to reality sought. That could include constraints such as specific accommodations like wheelchairs, vehicles starting from multiple depots, maximum total ride time of a vehicle and so on.

DARP can be divided into two cases: static and dynamic. The static case assumes that every request is known prior to the beginning of the day. This allows to a simple planning early in the morning. The dynamic case, on the other hand, describes situations where requests keep coming as the day goes. The planning must be adapted for every new request. The first one is, of course, simpler but might be less realistic.

Another division can be made between the multiple vehicles DARP and the single vehicle DARP. The latter being a particular case of the first.

For the remainder of this paper, we will focus on static multiple vehicles DARP

2.2.Literature review

The aim of this thesis being the creation of an algorithm solving static cases, I will focus on literature talking about those. However, a summary of the algorithm developed for dynamic case made by Cordeau and Laporte (2007) can be found at Appendix I. Furthermore, the aim of this literature review is to identify which types of algorithms are commonly used to solved DARP rather than the objective function and constraints usually employed. For that reason, I will focus on the evolution of the type of algorithm used. A detail review of the addition of new constraints or the modification of the objective's functions can be found in Cordeau and Laporte (2003a,2007) and in Parragh et al. (2008)

The first kind of algorithm that got developed was algorithms for single vehicle DARP which is a simplify version of the multi-vehicle DARP. The aim of those algorithms was to build a step into the solving of multi-vehicle DARP. To only mention a few, Psaraftis (1980 and 1983) developed two exact algorithms and Sexton (1979 and 1985a, b) two heuristics.

With the bases developed for the single vehicle DARP, Jaw (1984) developed one of the first heuristics for multiple vehicle DARP. He created an insertion heuristic which aims to insert requests initially classed based on their earliest possible pickup times into the best feasible place into a vehicle route. This was further discussed in Jaw et al. (1986).

Several similar algorithms were proposed in the following years improving it with a new technique called clustering. This technique groups requests to be served by the same vehicle prior to insertion. This includes the algorithms proposed by Dumas et al. (1989a), Desrosiers et al. (1991) and Ioachim et al. (1995).

Since then, several other insertion algorithms were proposed. I won't describe each contribution since they mainly concern the objective function and the constraints. However, it is interesting to mention that some papers tried successfully to use metaheuristics to improve the clustering process. For example, Rekiek et al. (2006) proposed the utilisation of Genetic Algorithm for clustering.

Even if insertion heuristics used to be the reference to solve DARP, it was quickly replaced by metaheuristics. The difference between heuristic and metaheuristic is that a heuristic only applies to one specific problem while a metaheuristic is a resolution method that can be applied to several problems with adaptations. The reason behind this trend change is that most metaheuristics have been developed during the second half of the 1980s and the 1990s. Ever since, they have been used with success to solve various optimisation problems. They manage to find good solutions within acceptable computational time. They work especially well for medium to large instances where exact algorithms have difficulties.

For all those reasons, most of the papers published since proposed to use metaheuristics. Insertion heuristics are, however, not forgotten. Various algorithm integrates insertion heuristics in metaheuristic in order to find optimal results.

For example, Parragh et al. (2010) used Variable Neighbourhoods Search with three types of neighbourhoods, Jain and Van Hentenryck (2011) a Large Neighbourhood Search with constraints programming, Braekers et al. (2014) a Deterministic Annealing algorithm and Gschwind and Gschwind and Drexl (2019) an Adaptative Large Neighbourhood Search.

However, the metaheuristic more frequently used are the following:

- Simulated Annealing with Colorni et al. (1996), Baugh et al. (1998) and Zidi et al. (2012),
- Genetic Algorithm with Uchimura et al. (1999), Jørgensen (2007) and Atahran et al. 2014),
- Tabu Search with Cordeau and Laporte (2003a), Aldaihani and Dessouky (2003) and Melachrinoudis et al. (2007).

Furthermore, several hybrid metaheuristics combining two different heuristics have been developed. Those hybrid versions have been proven to yield better results than the classic metaheuristics. Ho and Haughland (2004), Guerriero et al. (2013) proposed a hybrid of Tabu Search with Greedy Randomised Adaptative Search, Parragh and Schmid (2013) a hybrid of Large Neighbourhood Search and Variable Neighbourhoods Search and finally Masmoudi et al. (2016) a hybrid Bee algorithm with Simulated Annealing and Deterministic Annealing.

In parallel with the use of heuristics and metaheuristics, several exact algorithms with Branch and Cut have been proposed. Branch and cut algorithms solve problems by relaxing the constraints. It means that the problems are solved by ignoring constraints. Those constraints are then reinserted one by one. The main drawback is the computational time that can be high.

There are two types of Branch and Cut algorithms that had been used. Three-index formulation such as the algorithm presented by Cordeau (2006) and two-index formulation like the one proposed by Ropke et al. (2007) or more recently by Braekers et al. (2014).

It is also worth mentioning the Hyperheuristic developed by Urra et al. (2014). Their Hyperheuristic search among several low-level heuristics for good solutions rather than searching in the problem space.

A table to summarise this section can be seen at Table 1.

Number of vehicles	Type	Algorithm	Reference
Single	Exact	Dynamic programming	Psaraftis (1980 and 1983)
	Heuristic	Iterates between routing and scheduling phases	Sexton (1979 and 1985 a, b)
Multi	Exact	Branch and cut with two-index formulation	Ropke et al. (2007)
			Braekers et al. (2014).
		Branch and cut with three-index formulation	Cordeau (2006)
	Heuristic	Insertion	Jaw (1984)
			Jaw et al. (1986)
		Insertion with clustering	Dumas et al. (1989a)
			Desrosiers et al. (1991)
			Ioachim et al. (1995)
		Insertion with clustering through Genetic Algorithm	Rekiek et al. (2006)
	Metaheuristic	Variable Neighbourhoods Search with three types of neighbourhoods	Parragh et al. (2010)
		Large Neighbourhood Search with constraints programming	Jain and Van Hentenryck (2011)
		Deterministic Annealing	Braekers et al. (2014)
		Adaptative Large Neighbourhood Search	Gschwind and Drexl (2019)
		Simulated Annealing	Colorni et al. (1996)
			Baugh et al. (1998)
			Zidi et al. (2012)
		Genetic Algorithm	Uchimura et al. (1999)
			Jørgensen (2007)
			Atahran et al. (2014)
		Tabu Search	Cordeau and Laporte (2003a)
			Aldaihani and Dessouky (2003)
			Melachrinoudis et al. (2007)
	Hybrid metaheuristic	Tabu Search with Greedy Randomised Adaptative Search	Ho and Haughland (2004)
			Guerriero et al. (2013)
		Large Neighbourhood Search and Variable Neighbourhoods Search	Parragh and Schmid (2013)
		Bees algorithm with Simulated Annealing and Deterministic Annealing	Masmoudi et al. (2016)
		Bees algorithm with Deterministic Annealing	
	Hyperheuristic	Search among several low-level heuristic	Urra et al. (2014)

Table 1: Summary of literature review

The primary objective of this paper will be to develop a heuristic to solve homogenous static case of DARP with multi-vehicles and single depot. In order to do that, I first created an initial algorithm based on specific objective functions and constraints. On the basis of this algorithm, several possible improvements will be tested. Depending on their effects over the performance of the algorithm or his realistic aspect, they will be kept in the final version of the algorithm.

This thesis is organised as follows. Section 2 methodology will include 3 parts. The first one will be a problem formulation based on the objective function and constraints I chose to tackle. The second one will be a description of the algorithm implemented. This will also include the decision-making for choosing this algorithm. Finally, the different steps of the implementation will be presented in the third part. Results and improvement will be discussed in Section 3. For that, I will begin by presenting the results of the initial algorithm. Then several possible improvements will be proposed and their impacts will be compared to the results mention above. This thesis will be concluded in Section 4.

3. Methodology

This section will present the methodology I followed in order to develop the initial version of my algorithm. It is composed of three subsections. The first one explains my problem formulation. The second one the choice of the type of algorithm I chose to design. Finally, the last one present the implementation of this algorithm.

3.1. Problem formulation

Dial a ride problem can have several formulations depending on the situation the algorithm wants to solve. The exact name of the problem I chose to tackle is the homogenous static DARP for multiple vehicles with one depot. As seen in Section 1.1, the terminology static implies that every request is known prior to optimisation and multiple implies that I work with more than one vehicle. The only new aspect is the homogenous, which refers to a homogenous fleet of vehicles. That means that every vehicle is considered to be exactly the same.

The other particularities of this formulation are:

- How the time windows are handled
- Different maximum ride time for every request
- Number of vehicles is a cost to minimise
- Possibility to have more than one customer per request

This problem formulation is based on the one presented by Roepke during the CAOS[1] Seminar Series of 2005. It has been adapted to fit my objective function and constraints.

DARP is modelled on a graph with a set of nodes and arcs G = (N, A). The nodes N corresponds to the pickup locations P = {1, …, n}: the deposit locations D = {n + 1, …, 2n} based on n requests to be fulfilled and the depot. The possible journeys of the set vehicles K of capacity Q are then represented by arcs linking those nodes to each other's. Each arc between nodes has a specified cost c_{ij} and a travel time t_{ij}. Each node has three components. A desired time d_i, a number of customers to be served q_i and a maximum ride time $LMax_i$. The last two are the same for corresponding pickup and deposit nodes.

In addition to that, there is a constant cost per vehicle used and it is assumed that it takes 1 min to load or unload a customer.

[1] Copenhagen Algorithms and Optimization Seminars

All of the above give the following mathematical model:

Parameters:

- n: requests to be fulfilled composed of a pickup location and a deposit location
- P = {1, …, n}: pickup locations
- D = {n + 1, …, 2n}: deposit locations
- N= P ∪ D ∪ {0, 2n + 1} : all the nodes with 0 being the departure and 2n+1 the arrival point. They both correspond to the same depot.
- K: Set of vehicles
- Q: Capacity of a vehicle
- q_i : Amount loaded onto a vehicle at node i. For the pickups, this value is positive and for the deposits negative
- $LMax_i$: Maximum time for node i which is the same for i+n
- d_i : Desired time for node i
- c_{ij} : Cost of travelling from node i to j
- c_α : Cost per vehicle used
- t_{ij} : Travel time between node i and j

Variables:

- X_{ijk} : Binary variable equal to 1 if vehicle k travels directly between location i and j
- Y_k: Binary variable equal to 1 if vehicle k is used
- A_{ik} : Time at which vehicle k arrives at node i
- Q_{ik} : The load of vehicle k after visiting node i
- L_{ik} : Ride time of node i in vehicle k
- Del_{ik} : Delay from desired time for node i in vehicle k

Objective function:

Minimise: $\sum_{i\in N}\sum_{j\in N}\sum_{k\in K} X_{ijk}c_{ij} + Y_k c_\alpha + \frac{1}{2}\sum_{i\in P}\sum_{k\in K} Del_{ik} + 2\sum_{i\in D}\sum_{k\in K} Del_{ik}$ (1)

Contraints :

$\sum_{j\in N}\sum_{k\in K} X_{ijK} = 1 \quad \forall\, i \in P$ (2)

$\sum_{j\in N} X_{ijk} - \sum_{j\in N} X_{n+i,jk} = 0 \quad \forall i \in P, k \in K$ (3)

$\sum_{j\in N} X_{0jk} - Y_k = 0 \quad \forall k \in K$ (4)

$\sum_{i\in N} X_{i,2n+1,k} - Y_k = 0 \quad \forall k \in K$ (5)

$\sum_{j\in N} X_{jik} - \sum_{j\in N} X_{ijk} = 0 \quad \forall i \in P \cup D, k \in K$ (6)

$A_{jk} \geq A_{ik} + q_i + t_{ij} - M\left(1 - X_{ijk}\right) \quad \forall i \in N, j \in N, k \in K$ (7)

$Del_{ik} = A_{ik} - d_i \quad \forall i \in P \cup D, k \in K$ (8)

$L_{ik} = A_{n+i,k} - (A_{ik} + q_i) \quad \forall i \in P, k \in K$ (9)

$L_{ik} \leq LMax_i \quad \forall i \in N, k \in K$ (10)

$Q_{jk} \geq \left(Q_{ik} + q_j\right)X_{ijk} \quad \forall i \in N, j \in N, k \in K$ (11)

$Q_{ik} \leq Q \quad \forall i \in N, k \in K$ (12)

$MY_k \geq \sum_{i\in N}\sum_{j\in N} X_{ijk} \quad \forall k \in K$ (13)

$X_{ijk} \in \{0,1\} \quad \forall i \in N, j \in N, k \in K$ (14)

$Y_k \in \{0,1\} \quad \forall k \in K$ (15)

The main difference between my formulation and those usually presented in papers is the time windows. Usually, time windows are considered as a constraint to be respected. Instead of that, I work with relaxed time constraints through desired times. This allowed me to construct an algorithm which is less restrictive. However, time windows being a key component of the DARP, I did not drop them completely. I have just handled them indirectly through the delays. Several papers allow for the violation of time windows through a violation cost. I have done the same but on a broader scale.

The objective function (1) has 3 components: the distance travelled, the number of vehicles and the delays. The first one is a classic of any transporter problem, but the two others are specific to my formulation.

I chose to add the number of vehicles as a variable to minimise to add a managerial perspective. For company that offer services corresponding to DARP, each vehicle has an additional cost. Indeed, a driver is needed for each vehicle. Furthermore, it is safer for companies to avoid using the entirety of their fleet. This way, they can parry unexpected events such as cars breaking down. The cost c_α must be adapted depending on the situation. A tuning of this cost must be conducted to fit reality. This

tuning must be done in comparison with the cost of the distance and the delays. Indeed, the cost must set according to the delays and travelled distances a company accepts to pay in order to use one less vehicle.

The delays represent an inconvenience cost for the users. They are particularly important in my formulation because it is this cost which controls the respect of the time windows. I have divided this cost into two. The first one corresponds to the delays from departures while the ladder to delays from arrivals. The impact of the first one is minimised by dividing it by two while the other is amplified by multiplying it by 2. The aim of those modifications is to reflect a more realistic attitude toward delays. Indeed, customers are more eager to arrive at their appointments in time even though they might need to wait longer before departure than the other way around.

There are several groups of constraint depending on their purpose.

The first group define the structure of the routes. For that, the constraint (2) ensures that a request is served exactly once and the (3) that the pickup and delivery of this request is fulfilled by the same vehicle. The constrains (4) and (5) respectively guarantee that if a vehicle is used, he will start and finish at the depot. Finally, the constraint (6) imposes the flow conservation, which means if a vehicle enters a node, he must leave it.

The next group concern the calculation of the delays. Constraint (7) set the visit time of each node. This then use in (8) for calculating the delays by comparing it with the desired times.

The third group forbid violations of capacity and maximum ride times. For that, (9) and (11) set respectively the ride time of each request and the load at each node. Those are then controlled by (10) and (12). In literature, the maximum ride time is usually fixed for every request. I chose to have different maximum ride time to reflect a more realistic approach to the service quality asked by users. Indeed, a user desiring to travel 1 km will accept to spend less extra time travelling than a user travelling 50 km.

The last interesting constraint is the (13) which set Y_k= 1 if any request has been assigned to the vehicle k.

3.2. Algorithm description

As presented in Section 1.2, there is a new trend in literature toward hybrid metaheuristic. They are proven to yield better results than classic ones. That is why I chose to implement such algorithm. I chose to create an algorithm similar to the one presented by Masmoudi et al. (2016) with some differences. The algorithm is a hybrid Bee Algorithm (BA) with Deterministic Annealing (DA).

For that, I will start by presenting each of the two algorithms separately before describing their hybrid version. It is in this description that I will present the differences between the version I designed and the one proposed by Masmoudi et al. (2016). This section will, however, remain theoretical. The actual structure of my algorithm will be discussed in the next section. Once the different components of my algorithm are presented, I will explain the decision process I had to choose this algorithm.

3.2.1. Bee Algorithm

They are several versions of The Bee Algorithm presented in literature. We will focus our description on the method developed by Pham et al. (2005) and discussed in more detail in Pham et al. (2015).

Bee Algorithm is a swarm based evolutionary algorithm inspired by the behaviour of scout bees. Swarm based algorithms mimic natural phenomena in order to search for optimal solutions. In opposition of other direct search algorithms, swarm algorithms explore a population of solutions rather than to focus on only one solution to improve. There are several other swarm algorithms the more famous being the Ant Colony Optimisation (ACO) and the Genetic Algorithm (GA). The main difference between ACO and BA is that AOC tries to avoid moving too far from their initial solutions while BA does not have such limits. We have seen in section 1.1.2 that GA has been used by several authors to solve DARP. However, it has been proven that Bees Algorithm tends to yield better result than the GA.

The BA is inspired by the behaviours of bees' colonies looking for good food sources. A beehive is organised in a complex structure where bees are assigned to specific tasks. The task we are focusing on for the BA is the scouting for good food sources to dispatch harvesting bees. Due to the fact that the harvesting of pollen only occurs during a certain period, it is important that the harvesting be as proficient as possible.

When searching for good locations to harvest pollen, scout bees begin by searching in random areas. Locations are evaluated according to several factors such as the sugar percentage of possible flower patches. Once they found a good one, they come back to the hive and communicate the quality of the location found by a dance called 'waggle dance'. This dance communicates information such as direction, distance and quality of the possible food source. Based on the dances of the initial scouts, harvesting bees are dispatched to the possible food sources depending on the fitness of the source. The fitness being a combination of the quality compared to the distance of the source.

However, the work of the scout's bees is not over yet. The scout's bee will lead the harvesting bees to the areas previously found and then search around this area for possible new food sources.

Furthermore, new scout's bee will be sent to areas that were not explored yet. This process is then repeated.

The Bee Algorithm follows the same process. A population of initial solutions are evaluated according to their quality. Based on those, an extensive search is conducted in the neighbourhood of the best initial solutions. Furthermore, solutions with average qualities are also considered, but fewer efforts are put into them. This can be seen as central processing unit[2] (CPU) allocation based on the quality of the solutions. The allocation is done according to three groups: a group of good possible solutions on which a lot of CPU is allowed, a group of decent solutions which receives a bit of CPU and the other solutions with poor quality which receive no CPU. This is the theoretical principle, however, in practice, adding some randomness in the making of the three groups yields better results.

After the different searches, a new population is created containing the new solutions found along with newly generated ones. The process is then repeated. The search for possible improving solutions made during the algorithm is done through local searches in the neighbourhood of the solution.

BA is actually combination of iterative local search and random global search. The first being the search in the neighbourhood and the second refers to the use of populations. This allowed easy hybridisation as the iterative local search can be done with other metaheuristics.

3.2.2. Deterministic Annealing

The Deterministic Algorithm (DA) or Threshold accepting was first introduced by Dueck and Scheuer (1990). It is a variant of the Simulated Annealing which is inspired by the annealing process used in metallurgy to obtain good metal quality. In this process, warm metal is cooled before being heated again in order to improve its overall quality. Adapted to an optimisation process, it means that a solution is worsened in order to find a better one at the end.

The DA is actually a simplification of the classic Simulated Annealing (SA) which has the advantage to be faster while keeping relatively good precision. However, DA does not guarantee to find the global optimum. Another advantage of the DA is that it is relatively easy to understand and implement. Furthermore, fewer parameters are needed.

DA is a meta-heuristic based iterative local search. At each iteration, local search operators are applied to a solution. Each time, the solution is improved thought the local search, it is accepted as the new current solution. Otherwise, it is accepted as long as it does not deteriorate too much the solution. For that, an acceptance threshold is defined. However, this threshold will evolve as the iterations go. Each time, a worse solution is accepted, the threshold is lowered until only improving solutions are accepted.

[2] computational time

3.2.3. Hybrid Bee Algorithm with Deterministic Annealing

A hybrid Bee Algorithm with Deterministic Annealing has several advantages. The BA hybridised well with DA because the DA can easily replace the iterative local search of the BA. Furthermore, DA has the inconvenience to not guarantee global optimum. This is solved through the random global search of the BA. Finally, the speed of the algorithm remains reasonable thanks to the fact that DA is quite fast.

As mentioned before, the hybrid BA with DA I have chosen to implement is derived from the one proposed by Masmoudi et al. (2016). In this paper, three metaheuristics are presented to solve DARP. An Adaptative Large Neighbourhood Search, Hybrid Bee Algorithm with Deterministic Annealing and Hybrid Bee Algorithm with Simulated Annealing. Their study shows that the two hybrid algorithms outperform the classic Adaptative Large Neighbourhood Search or even simple BA, DA or SA. Furthermore, BA with DA works faster the BA with SA while keeping the same precision.

There are several differences between the BA with DA presented by Masmoudi et al. (2016) and the one I made. They once again concerned the objective function and the constraints, but also technical elements of the algorithm structure.

Masmoudi et al. (2016) worked on multi-depot multi-trip heterogeneous dial a ride problem while I work on single depot single trip homogenous dial a ride problem. This means that I work with only one depot with one trip instead of multiple depots with multiple trips due to lunch and coffee break. Furthermore, my fleet of vehicles is homogenous instead of heterogenous meaning all my vehicles are the same while they deal with vehicles with different accommodations and capacities.

The next difference is the control of the time windows. Indeed, I chose to tackle this constraint as desired time and delays while they kept it as a classic constraint.

Next, I added the number of vehicles as a variable to minimise. I also specified different maximum ride times for each request depending on the distance they wish to travel instead of a constant maximum ride time for every user.

Algorithm by Masmoudi et al. (2016)	My algorithm
Objective function	
Multiple trips to multiple depots	Single depot with one trip
No delay minimisation	Minimisation of delays
No vehicle minimisation	Minimisation of vehicle
Constraints	
Lunch and coffee breaks	No breaks
Multiple type of vehicles	One type of vehicle
Multiple trips per vehicle	One trip per vehicle
Time windows	Desired times
Constant maximum ride times	Adapted maximum ride times

Table 2: Algorithm comparison

On a more structural level, I have implemented different local operators than them and my solution creation process differs for the one they proposed. I also adapted the parameters and stopping criterion while remaining in the same ideas.

Finally, the last big difference is that I have added several iterations of a large search operator in order to improve my solution even more at the end of the classic hybrid BA with DA.

3.2.4. Decision-making

After my literature review, it was clear that hybrid metaheuristics perform better than classical ones. I just had to choose which hybrid version I would implement. I initially wanted to create a hybrid version of the Genetic Algorithm but after some research I realised that the BA is proven to yield better results while keeping the same logic.

Once I had decided to use a BA, I still needed to choose which other meta-heuristic to hybridise it with. The paper of Masmoudi et al. (2016) gave me a good comparison between two algorithms I was already considering which are DA and SA. I initially wanted to develop a hybrid with SA but their paper showed me that DA was not only faster but also simpler to understand and implement. The speed of the algorithm was a decisive factor because my computer is not very performant. That is why the balance between computational time and precision was particularly important in my decision-making process. Furthermore, DA has fewer parameters than SA which means fewer to tune.

For all those reasons, the hybrid BA with DA was the perfect compromise as it was fast while having the same level of precision of the hybrid BA with DA.

However, I still wanted to improve the precision. That is why I added the Large Neighbourhood search to my algorithm.

3.3. Algorithm implementation

In this section, I will describe how I implemented my algorithm. For that, I will start by an overlook of the algorithm before describing its components.

3.3.1. Algorithm structure

The structure of the algorithm can be seen in the Algorithm 1 and the flowchart of Fig3. However, some explanations are needed. In this explanation, I will use the term current best solution and global best solution. The current best solution refers to the best solution in a specific iteration. On the other hand, the best global solution refers to the best solution found so far. In addition to that, the notation N, be, es will be used several times. Those notation both refer to groups and the sizes of those groups.

Algorithm 1: Pseudo-code of the proposed hybrid Bee Algorithm with Deterministic Annealing

```
START
  Creation of initial Population N
    REPEAT
      Step 1: Evaluate the quality of each solution of population N and memorise the best current
      one
      Step 2: Select (be) solutions from N and sort them in ascending order
      Step 3: Select the best (es) solutions from the (be) solutions
      Step 4: Apply Deterministic Annealing on each of the (es) solutions
      Step 5: Apply few iterations of local search operators on the (be-es) solutions
      Step 6: Evaluate the quality of the new solutions and select the best one
      IF the current is the better than the best global
            THEN Replace the global best with the current one
      END IF
      Step 7: Insert the (be) new solutions in the population N
      Step 8: Create (N-be) new solutions
    UNTIL Stopping criterion reached
    REPEAT
      Step 9: Apply Large local search operator on the best solution
    UNTIL No improvement after 5 consecutive applications
  OUTPUT Best solution
STOP
```

The algorithm starts with a constructive insertion heuristic to create the initial population N. After that Step 1 to 8 are repeated until the stopping criterion is reached.

The Step 1 evaluates the solution of the population N. At the first iteration, the best solution will be memorised as the best global solution. For other iterations, it will only be memorised as the best current one.

A selection is made among the population N. The selected (be)[3] solutions from N are then sorted in ascending order. Those two processes constitute the Step 2.

The best (es)[4] solutions from (be) solutions are selected in Step 3 and Deterministic Annealing is applied to them in Step 4.

The other (be-es)[5] solutions go through a local search for few iterations in Step 5.

The Step 6 is similar to the Step 1 except on the newly found solutions of (be).

Step 7 insert the (be) new solutions in the population N and finally the rest of the population is replaced with new solutions in Step 8.

To finish the algorithm, iterations of large neighbourhood search is applied in Step 9 until there is no new improvement for 5 consecutive iterations. The result gives us our best solution.

The sizes of the groups N, (be), (es) and (be-es) depend on parameters of the same names.

This algorithm[6] will be implemented in Julia Programming Language[7] and the tests are performed on a laptop Lenovo Y700 with Intel® Core ™ i5-6300HQ CPU @ 2.30 GHz and 8 GB of RAM. The laptop is, however, old thus his performances are weakened.

[3] Group of solutions to be explored
[4] Group of solutions to be explored in depth.
[5] Group of solutions to be explored succinctly
[6] The complete code can be found at:
https://drive.google.com/drive/folders/1yxdy5vMW6WKlGJRBcO4SW4clfQa8ywsb?usp=sharing
[7] https://julialang.org/

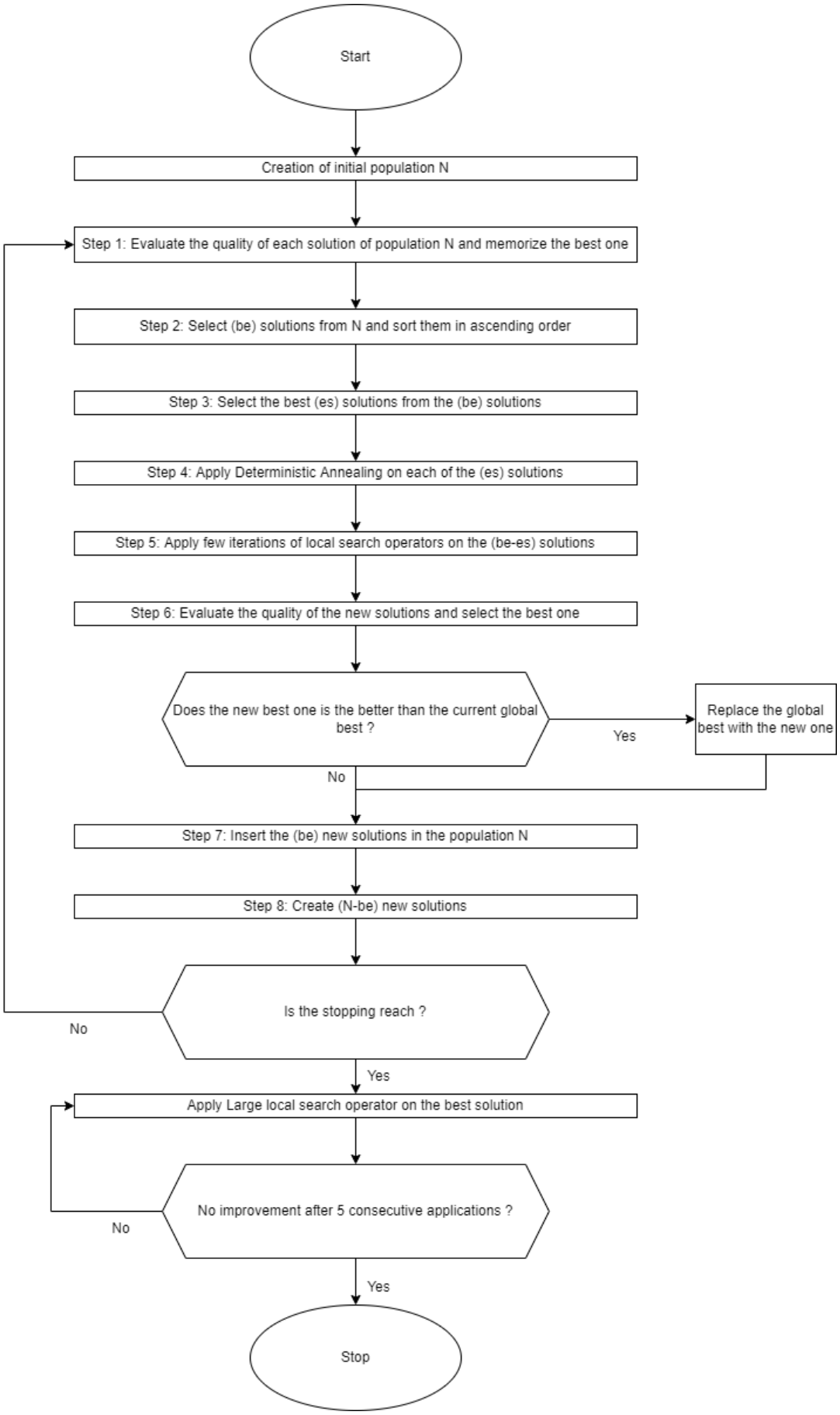

Fig 3. Flowchart of the hybrid Bee Algorithm with Deterministic Annealing

3.3.2. Components

3.3.2.1. Instances

Due to the versatility of the problem regarding his objective function and his constraints, there are a lot of different instances proposed in literature. Papers usually agree on the instances proposed by Cordeau and Laporte (2003b) as a basis to build instances adapted to one's problem formulation.

Nevertheless, I have decided not to use those and instead chose the one proposed by Chassaing et alt (s.d). Their instances followed the structure of the ones proposed by Cordeau and Laporte (2003b) but are more realistic. Indeed, it is possible to have more than one customer per request and the distances are calculated based on the real distances that a vehicle would have to make rather than on the Euclidian distances between the two points. This means that triangular inequality is not respected. Those instances only take into account one type of customer. Furthermore, each request has a maximum ride time assigned based on the distances he wants to cross rather than a constant maximum ride time for every request.

They also have the following specificities:

- The points refer to cities in France.
- The vehicles are believed to travel at a constant speed of 80 km/h.
- The requests of each instance are dispatched on territory corresponding to a French department. This means areas vary between 100 km² and more than 8500 km².
- Distances are given in metres.
- Time windows are given in minutes.
- The n requests are divided into pickup and delivery.
- A number from 1 to n is attributed to the pickup.
- The deliveries are given the number corresponding to their pickup +n.

I still had to adapt those instances to my constraints. For that, I have transformed the time windows into desired delivery and pickup times by taking the median of the time windows. For example, a node with a time window of [480,650] will have a desired time of 565[8].

Finally, the speed was given in kilometres per minute, so a basic conversion was made to transform it in metre per minute. In the rest of the algorithm, everything will be handled either in minutes or metres.

The size of the instances can be very different. In order to have a better understanding during the tests, I have divided them into 3 subgroups depending on their sizes. Instances with under 50 requests are considered to be small, instances with 50 to 75 to be medium and the one with over 70 requests to be large. The range for the medium instance seems small, but in practice the majority of the instances are medium.

[8] (480 + 650) /2 = 565

3.3.2.2. Solution structure

The solution structure is a key element of the algorithm since every other component will be constructed to work with this structure. That is why, a simple combination of numbers representing the request is not enough. This structure must contain more than that. Key variables must be kept in my solution structure and evolve as the solution evolves in order to avoid calculating them again at every step they are needed.

Among the six variables I have presented in section 5.1, I have chosen 3 of them who will be a part of my solution structure:

- The time at which a vehicle arrives at a node
- The load of the vehicle after visiting a node
- The delay between desired time and actual time of arrival

The two binary variables are, of course, indirectly represented as they define the journey of a vehicle and the allocation of requests to those vehicles. Finally, the ride time of a node will not be a part of the solution. This will only be used in the feasibility check of my solution thus will be calculated when needed.

In addition to that, I have decided to keep the desired departure times of each node as this data will be used at different steps of the algorithm.

The actual structure is a vector of vectors of vectors. The first level has as much vector as they are vehicles used in the solution. This level represents the solution as an all. The second one has as much as there are nodes in a vehicle trip. It represents the sequence a specific vehicle follows. Finally, the last level represents each node with the data associated to it. For this level, the data are presented in the following order are [Node number, Desired Time, Actual Time, Delay, Number of users inside the vehicle]. The desired time, the actual time and the delays are all in minutes.

This gives us solutions like the one presented in Fig.4.

[13 360 360 0 3; 14 360 380 20 6; 18 360 378 18 7; 28 390 390 0 8; 42 409 415 6 5; 46 415 443 28 4; 25 420 480 60 8; 41 456 456 0 5; 53 489 489 0 1; 56 491 491 0 0; 11 570 570 0 2; 39 626 626 0 0; 4 630 636 6 4; 32 672 672 0 0; 15 810 810 0 1; 43 851 851 0 0; 1 1080 1080 0 4; 29 1103 1103 0 0]

[23 390 390 5 3; 51 444 444 0 0]

[9 360 360 7 4; 3 360 421 61 7; 31 411 411 0 4; 37 439 439 0 0; 8 600 600 0 3; 36 663 663 0 0; 10 840 840 0 2; 27 840 935 95 6; 38 947 947 0 4; 55 985 985 0 0]

[22 330 330 0 1; 5 360 360 0 2; 7 360 384 24 6; 50 397 398 1 5; 33 416 416 0 4; 35 421 448 27 0; 26 600 600 0 2; 54 658 658 0 0; 2 660 676 16 1; 30 800 800 0 0; 20 840 842 2 1; 21 870 870 0 2; 48 969 969 0 1; 49 975 997 22 0]

[19 330 330 0 4; 17 360 364 4 5; 24 360 399 39 8; 52 420 420 0 5; 47 428 447 19 1; 45 464 470 6 0; 12 570 570 0 4; 40 619 619 0 0; 6 660 670 10 3; 34 758 758 0 0; 16 870 870 0 4; 44 932 932 0 0]

Fig 4. Solution example

This solution has 5 vehicles within []. Nodes inside a vehicle are separated by ; .If we take a closer look at the fifth vehicle, we can see that the vehicle will begin by going to the node 19 at the desired time 330 to pick 4 customers. After that he will go to the 17 at 364 with 4 min of delay from desired time to pick 1 customer and so on. If we focus on the node 19, we can see that the delivery of this node will be the fifth stop as there are 28 requests[9].

A clearer representation of the journey of vehicle 5 can be seen in the Fig. 5 with the numbers over the cases being the number of customers entering or leaving the vehicle at this node and the numbers under being the time of arrival at a node.

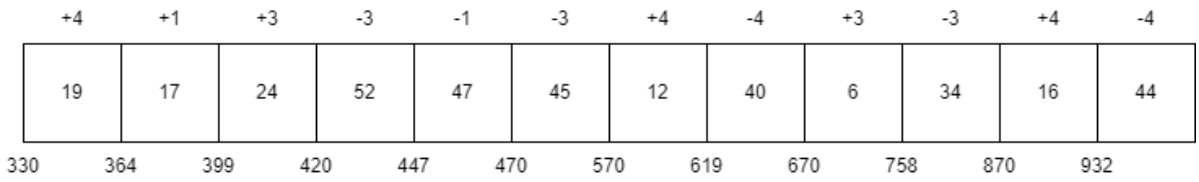

Fig. 5: Example solution representation

3.3.2.3. Feasibility check

The feasibility check controls the constraints 9 to 12 of my problem formulation. It makes sure that the capacity of a vehicle is never passed and that the maximum ride time of each request is respected.

This feasibility check is done at several points of the algorithm. The first one being, of course, as a solution is created and the other points are at each local search, since each local search operator will create new solutions.

However, it will not be the same at every point. Indeed, enforcing the respect to these two constraints at every point of the algorithm restrict the diversity of the solutions explored. The solutions created are too similar if they have to respect both constraints. This can lead to the algorithm sticking itself into local minima.

That is why a relaxed feasibility check is performed during the hybrid DA with BA and a restrictive on the final solution. This relaxed version only force the respect of the vehicle's capacities and allows for violation of maximum ride times. This is designed this way as it is more realistic to let users ride longer than planned rather than allowing having more people than places in a vehicle.

Nonetheless, the maximum ride times are not removed completely from the algorithm. Instead, each violation is penalised by a huge fee. This way, several solutions with cost violations are created during the different stages of the algorithm. However, the final solution never has any violations.

[9] 19 + 28 = 47

3.3.2.4. Calculation cost

The cost calculation will, of course, be as describe in the objective function presented in 2.1. However, two points need to be explored in detail.

The first one concerns the cost per vehicle used. A possibility would have been to have hierarchical objective function. Which would have meant that the algorithm would prioritise the minimisation of an element of the objective function before looking to minimise the other ones. In my case, I would have looked to minimise the distances then the delays and finally the number of vehicles.

Instead of doing that, I chose to convert each element of my objective function into the same unit: an amount of time. This allows me to have a single cost to minimise without having to hierarchies them. For that, the distances are converted into travel times thanks to the speed of the vehicle. The delays already are represented by amounts of time and finally, the cost per vehicle is a constant amount of time. This constant amount of time would need a lot of tuning in order to represent the actual cost of a vehicle to a company. However, this will not be explored in more detail here as it is not the aim of this thesis. In this algorithm, it will be equal to twice the travel time between the depot and the farthest node.

The second point concern the maximum ride times. As we have seen in the previous section, the feasibility check allows the violation of this constraint at the cost of a huge fee for intermediary solutions. This fee is similar to the cost per vehicle with the exception that it is also multiplied by the number of vehicles available. This way maximum ride time violation will only occur if no empty vehicles are available.

3.3.2.5. Solution creation

To create my solutions, I used the insertion heuristic presented by Jaw (1984) and further discussed in Jaw et al. (1986) with some modifications. The idea behind it is to insert requests one by one inside the solution at the best feasible places.

Jaw (1984) presents this insertion heuristic on two DARP. The first has relaxed time constraints with desired times and the second stick time constraints with time windows. The first case corresponding to my problem formulation, I will focus on it.

I still made two important changes from the version of Jaw (1984). The first one concerns the order in which requests are inserted into the solutions. Jaw develops several techniques such as sorting in order to improve the final solution of his heuristic. However, the quality of the solution is not really the aim of this process. I want to have a diversified population of solutions with decent quality not optimal quality. That is why, the requests are inserted in random order into my solutions. This way, I obtain a new solution each time the process is used.

The second change concern the actual insertion of a request inside a vehicle. Jaw explores each feasible way a request can be inserted inside a vehicle and then select the one with the lowest cost. Instead of doing that, each time I insert a request into a vehicle, she is put in the first place. Following that, the nodes of the vehicle are sorted according to their desired time. With this method, the sequence of a vehicle might be suboptimal. That is why a local search will be developed later to rearrange the sequence of a vehicle in order to find a better one.

This second change was made in order to accelerate the solution creation process which will occur at each iteration of my algorithm.

Each time a population is created, they will be evaluated according to their cost.

3.3.2.6. Selection phase

The selection phase is done to form the groups of solutions (be) and (es).

From the initial population, a predefined number of solutions are selected to compose the group (be). Those selected solutions are then separated into two groups based on their qualities. For that, the group (be) is sorted in ascending order. The first group correspond to the (es) solutions. Those solutions will be explored in depth in the following section. On the other hand, the second group of (be-es) solutions will be explored succinctly.

It could be reasonable to believe that the tournament process should always select the best solutions of the current population. However, that would not lead to the best result because it limits the diversification of the algorithm.

The selection process is a key element of all the metaheuristics based on population. For that reason, several techniques exist. Those techniques can be divided into two types: proportionate-based and ordinal-based. The first select solution based on calculated probabilities while the latter select based on an order. An extensive list of techniques has been presented by Miller and Goldberg (1995).

Two techniques have been explored, one of each type. In both the quality of a solution refer to his cost.

The first technique is proportionate-based. The principle is that each solution has a probability to be selected according to the relative difference between the worst solution of the population and them. What this mean is that the proportion of a solution is higher if his relative quality is better.

The formula used is the following:

$$Prob_i = \frac{Cost_{MAX} - Cost_i}{\sum_{j \in N(} Cost_{MAX} - Cost_{j)}} \quad \forall i \in N$$

$Cost_{MAX}$ being the cost of the worst solution of the population.

This selection is repeated (be) times.

The second technique is called tournament selection. It was proposed by Miller and Goldberg (1995). It has been tested for several transportation problems by Freitas (2013) with good results.

This technique works by randomly selecting a subgroup of the population of size (s). Then the solution of the subgroup with the best quality is selected. The process is then repeated (be) times.

After some experimentation, I have realised that the first method led to the same solution being selected over and over. This led to a fast convergence of my population. That is why, the second technique is the one that I have used in this algorithm.

It is theoretically possible that the current best solution of the algorithm is lost during this phase. This is due to the random factor of this selection process. This is attended as this current best might be a local minimum. That is why exploring other solutions might yield better result than exploring this global best solution at each iteration.

3.3.2.7. Improvement phase

The improvement phase is where the algorithm search for better solutions. While it is possible to find good new solutions during the solution creation process, the chances are low. That is why this phase is the key of this algorithm.

This section will include both the Deterministic algorithm and the local search. They are applied in parallel on the (es) solutions and on the (be-es). I will start by the DA as it is the more important one. These two techniques will be followed by a check for a new global best solution.

a) Deterministic Annealing on the (be) Solutions

Where classical Bee Algorithm would only make a local search on this level, I have chosen to implement a Deterministic Annealing as it worked well as an improvement metaheuristic. This DA will be applied on each of the (es) solutions.

The main component of the DA is the threshold T, which will determine if a solution found after the application of a local operator will be accepted or not. It will be accepted if the solution is feasible and the worsening of the solution after the application of the local operator does not exceed the current threshold. Of course, if a solution is improved, she will be accepted directly. Each time an iteration is concluded without improvement of the best solution, the threshold T is reduced by predefined amount ΔT. T decrease until no worse solutions are accepted.

Furthermore, if the threshold become negative, it is reset to a certain value.

This DA will be based on the one presented by Braekers et al. (2014). The main difference will be the stopping criterion I used. Braekers et al. (2014) stopped their algorithm after 25000 iterations. However, this DA being applied to several solutions at each iteration of my algorithm this number must be tuned down. Masmoudi et al. (2016) tune this number iteration down to 5000. This remains too high for my algorithm as each iteration would go on for at least 15 minutes.

Several numbers of iterations were tested but finding a number of iterations which keep acceptable time for each instance is not an easy task. Indeed, large instances may take 15 minutes to perform an iteration while a small instance takes 15 seconds.

For that reason, I completely change the stopping criterion. Instead of it being a defined number of iterations, it is a defined amount of time. As long as the amount of time defined is not passed, the DA will start a new loop. This way, I maintain an acceptable run time for every instance. This, however, led to a drop in precision as the size of the instance increase. The stopping time chosen is 15 seconds.

A pseudo-code of my DA can be seen in Algorithm 2. His functioning will, however, be discussed here.

Before starting, the threshold T is set to his maximum T_{max}, i_{imp} to 0, X_{Best} and $X_{current}$ to be the initial solution X_{ini}. The following steps are going to be repeated for a predefined amount of time. At each iteration, 1 is added i_{imp}. This i_{imp} will monitor the number of iterations since an improvement of the global solution is found. Each time a better global solution is found, it is set back to 0.

At this point, each local operator is applied to $X_{current}$ in random order. Each time an operator is applied, we checked if the new solution X_{mod} is accepted. This requires two things. First, the solution must be feasible. Second, the solution must either be an improvement of the X_{Best} at which point X_{mod} become the new X_{Best} and i_{imp} set to 0 or be accepted through the threshold T.

Once each operator is applied, the second part of the algorithm begins. This part will have two purposes. Adapt the threshold and avoid that the algorithm goes too far from the X_{Best} without any success. Each time an iteration is done without any improvement of X_{Best}, ΔT is subtract from T. After a certain number of iterations, T will become negative. When it arrives, it is reset to proportion of T_{max} calculate by multiplying T_{max} by a random number between 0 and 1. Each time this is done, we check the number of iterations since an improvement has been found thanks to i_{imp}. If the number of iterations is bigger than the predefined number n_{imp}, we reset $X_{current}$ to the value of the best solution found so far X_{Best}.

Algorithm 2: Pseudo-code of the proposed Deterministic Annealing

START
  Initialise Threshold $T = T_{max}$, $i_{imp} = 0$ and $X_{Best} = X_{current} = X_{ini}$
    REPEAT 1
      Add 1 to i_{imp}
      REPEAT 2 for each operator
        Apply a local search operator L on $X_{current}$ to obtain $= X_{mod}$
        IF 1 X_{mod} is accepted
          THEN $X_{current} = X_{mod}$
          IF 2 the cost of $X_{current}$<cost of X_{Best}
            THEN $X_{Best} = X_{current}$ and $i_{imp} = 0$
          END IF 2
        END IF 1
      END REPEAT 2
      IF 3 i_{imp}>0
        THEN subtract ΔT from T
        IF 4 T <0
          THEN Set $T = T_{max} \times random\ number\ between\ 0\ and\ 1$
          IF 5 $i_{imp} > n_{imp}$
            THEN $X_{current} = X_{Best}$ and $i_{imp} = 0$
          END IF 5
        END IF 4
      END IF 3
      UNTIL 1 The maximum amount of time is passed
  OUTPUT X_{Best}
STOP

This process is time consuming but yield good results. That is why it is only applied to the (es) solutions and not on the entirety of the population (be).

b) Local search on the (be-es) solutions

The solutions (be-es) having worst qualities than the (es) solutions, the local search applied on them will be more succinct. To each of them, four local search operators are applied in random order for a predefined number of times n_{max}. Each time a local search operator is applied to a current solution, the newly constructed solution is kept only if it is feasible and its cost is better than the current solution.

This method is simpler and solutions are more likely to be stuck into local minima. However, applying this local search improve the global performance of the algorithm since global best solutions can be found in this process.

c) Check for new global best

Now that the selected solutions (be) have been improved. I can check if the improvement phase has found a new global best solution. For that, a comparison is made between the current best solution of (be) and the global best.

3.3.2.8. Population adaptation

Once the improvement phase is over, i need to diversify our population in order to avoid a convergence. For that, a new population must be created.

To create this population, i first keep all the (be) solutions that i have improved during the improvement phase. To fill the rest of the population there are two possibilities. Either I keep the other solutions of the initial population and modify them or create new solutions completely. I have chosen the latter.

3.3.2.9. Stopping criterion and large operator

The processes describe the three previous sections are then repeated until our stopping criterion is reached. The stopping criterion I chose is to monitor the number of iterations since my hybrid BA with DA has found a new global best solution. If no better solution has been found for five consecutive iterations, the BA is stopped.

On this global best, I will conduct an additional large neighbourhood search. For that, I use a specific large search operator different from the ones used during the BA.

This large neighbourhood search is conducted as long as better solutions are found. The solution obtained is the final solution.

3.3.2.10. Parameters

One of the drawbacks of the utilisation of a hybrid algorithm is that it increases considerably the number of parameters to tune. Indeed, each meta-heuristic has several parameters that need to be adapted to each problem but also to the set of instances used. In addition to that, I also have an additional parameter (s) which comes from the tournament selection. The parameters I chose to apply were based on recommendations of literature mixed with some intuitions. I will present them in the following order: the parameters of the BA, the parameter from tournament selection and finally the parameters of the DA. The parameters of the BA having direct influences over the others, they will be presented first.

a) Bee algorithm

There are four parameters for the BA. They determine the size of the different groups used during the algorithm. In order to set them, I used the computational result of the parameters setting of the algorithm of Masmoudi et al. (2016) (Appendix II).

The first one is the size of population N, the second the size of the test population (be) and the last the size of the group for deep investigation (es). Their results show that they obtain a good compromise between precision and computation time by setting N = 20, be = 15 and es = 5. However, in order to save computation time, I chose to set them to N = 10, be = 5 and es = 3. Indeed, with those parameters, the computation time drops significantly without degrading to much the precision. They need 2/3 of the time for a slight drop in precision.

A fifth parameter can be considered, the number of iterations of local search done on the (be-es) solution. For this one, I used the same as Masmoudi et al. (2016) which is 10.

b) Tournament selection parameter

The parameter (s) of the tournament selection is the parameter that decides how many solutions are taken to form a subgroup from which the best solution is selected to be a part of (be). The bigger this number is, the higher chance there is that the same solutions are selected more than once. This parameter directly depends on the size of the population N decided above.

Tests have been conducted with the value 2,3,4 and 5. During these tests, I monitored the evolution of the population and realised that the value 3 allowed for a good level of diversification without losing too many good solutions.

c) Deterministic Annealing parameters

DA has, by far, the hardest parameters to set. Indeed, the different parameters directly interact and the impact of the modification of one of them has consequences to all the others. Furthermore, the stopping criterion directly impacts the setting of the parameters. In order to set them, I took elements

from Masmoudi et al. (2016) and Braekers et al. (2014). However, the stopping criterion of both their algorithms being considerably bigger than mine. That is why I had to tune them down following my intuition.

The three parameters to set are the maximum threshold value T_{max} , the threshold reduction ΔT and finally the restart parameter n_{imp}. The first two are directly related as ΔT can be seen as the proportion of T_{max} to withdraw each time no improvement has been found. In addition to that, the last two are related to the number of iterations the algorithm performs.

The threshold must be reduced fast enough that at several iterations, only improving solutions are accepted but not too fast that not enough worst solutions are accepted. The cycle of acceptance and restriction of the threshold must happen a correct amount of time during the number of iterations the algorithm performs. Furthermore, as n_{imp} decides the number of iterations before resetting the solution to the best found so far, it must also be set according to the number of iterations. This reset must happen but not too often.

The problem is that, I have chosen to use an amount of time as a stopping criterion. Because of that, I cannot directly influence the number of iterations. That is why in order to set the different parameters I monitored the number of iterations that could be performed in my specified amount of time. As one can suppose, this number varies a lot depending on the size of the instances. I still needed to find a good compromise that would be decent for every instance. After some testing over several instances, I decided to assume an average number of iterations of 150.

For T_{max}, I used a technique presented by Braekers et al. (2014). This technique is to use a relative T_{max}, which is defined as the average distance between two locations multiplied by a new parameter t_{max} to be set. This t_{max} is set to 1.2 as recommended by Braekers et al. (2014).

For n_{imp}I used the exact same n_{imp} as Masmoudi et al. (2016). It is equal to five times the number of vehicles currently used in the solution.

It is for ΔT that I had the most problem setting. Braekers et al. (2014) used 300 and Masmoudi et al. (2016) a proportion of T_{max} which is 1/2500 of T_{max}. I followed the second method as the first seemed to be dependent on the instances and problem formulation. I still had to adapt it to my algorithm as this proportion was appropriate for them because Masmoudi et al. (2016) had a number of iterations of 5000. Their proportion being one over half their number of iterations, I decided to take ΔT= 1/75.

3.3.2.11. Local operators

The local search operators might be the most important part of my algorithm. They are used in three different points: in the Deterministic Annealing, in the local search and in the large neighbourhood search. Furthermore, they are the parts which takes the most time to develop as they modify solutions directly.

Seven operators have been implemented and tested. However, only five of them are kept in the algorithm. Furthermore, three of them are similar. For that reason, I will begin by presenting those three. After that, I will present the two other used ones and finish with the two I decided to drop.

For each of them, a pseudo-code will be given and a feasibility check is performed on each new solution.

a) Relocate

This operator is the basis from which the next three are inspired. The idea is to search for which request will benefit by being relocated into a different place of the solution. For every request, we remove them from the solution and replace them as the best feasible place. We then memorise this new solution. Once this is done for every request, we look among all the newly generated solutions which is the best, which will be the output of the operator.

Algorithm 3: Pseudo-code of operator relocate

```
START
        REPEAT 1 for each request
                Remove the request for the solution
                REPEAT 2 for each vehicle
                        Insert the request at best place inside the vehicle
                END REPEAT 2
        END REPEAT 1
        Check the costs of the solutions created above and keep the best
   OUTPUT Best relocation
STOP
```

Despite working quite well, this operator has a huge drawback. He is really time consuming. He is too wide to be used inside the DA. Another version of this operator that keeps the same idea but works differently have been tested. This version memorises only the best relocation found so far to avoid the final check. However, this change does not improve the speed of the algorithm enough.

For that reason, two similar operators have been designed to replace him. For each of those derived operators, a component will be removed from the search.

On the other hand, this operator will be used in the large neighbourhood search.

b) Relocate 1 to 1

This is the first operator created to replace the initial relocate. The component that has been removed is the selection of the vehicle. Indeed, it focuses on 2 vehicles rather than all of them. Instead of looking each request, the operator select randomly 2 vehicles. It moved a request from the first vehicle to the other. The request to be moved is the one for which the relocation has the best impact on the cost of the solution.

Algorithm 4: Pseudo-code of operators relocate 1 to 1

```
START
        Select 2 vehicles randomly
        REPEAT for each request of the first vehicle
                Move the request to the best possible place of the second vehicle
        END REPEAT
        Check the costs of the solutions created above and keep the best
   OUTPUT Best relocation
STOP
```

This operator is currently used both in the DA and the local search on the (be-es) solutions.

c) Relocate 2 to any

This is the second operator derived from the initial relocate. This one focus on 2 requests rather than all of them. For that, 2 requests are randomly selected. They are then removed from the solution and replace at the best possible place in each vehicle. This, of course, excluding the vehicle they were initially. The relocation with the best impact over the global solution is kept.

Algorithm 5: Pseudo-code of operators relocate 2 to any

```
START
        Select 2 requests randomly
        REPEAT 1 for each of the two requests
                Remove it from the solutions.
                REPEAT 2 for each vehicle
                        Insert the request at best place inside the vehicle
                END REPEAT 2
        END REPEAT 1
        Check the costs of the solutions created above and keep the best
   OUTPUT Best relocation
STOP
```

This operator is currently used both in the DA and the local search on the (be-es) solutions.

d) Swap

The idea of this operator is to exchange two requests from two different vehicles. In order to do that, two vehicles are selected randomly. Then for each request of the first vehicle, we search for the request from the second vehicle with the most similar desired times and exchange the two. The exchange with the best impact over the global solution is kept.

Algorithm 6: Pseudo-code of operator swap

```
START
        Select 2 vehicles randomly
        REPEAT for each request of the first vehicle
                Check all the requests from the second vehicle to see which one has the more similar
                desired times
                Exchange those 2 requests
        END REPEAT
        Check the cost of the solutions created above and keep the best
   OUTPUT Best swap
STOP
```

This operator is currently used both in the DA and the local search on the (be-es) solutions.

e) Cut and reform

This operator is the most complex. What it does is cut the sequences of two vehicles. The beginning sequence of the first vehicle is then reattached to the end sequence of the second vehicle and vice versa. The cuts cannot be done at any places or the pickup and delivery of a request might find themselves into separate vehicles. This would not make any sense. That is why cut can only be made at moments when the vehicle is completely empty.

In order to do that, a vehicle is selected randomly. Each possible cut in the sequence of the vehicle is identified by looking the evolving capacity of the nodes. If a node has an evolving capacity of 0, a cut is possible as it means the vehicle is empty until he takes new users. Once this is done, we have a list of possible cuts for our vehicle. The next step is looking for cuts in the other vehicles. Each vehicle is considered. Each one is scanned for possible cuts. The local operator will now take each vehicle one by one. He searches between the cuts of the initial vehicle and those of other vehicles to find pairs of cuts with the highest resemblance. The resemblance is based on the actual time the node is processed. Those pairs of cuts will be used to form the new solution.

For example, if a solution has 6 vehicles, the operator finds 5 pairs.

The sequences of the two vehicles are cut according to the pair finds above. Then, the beginning sequence of the first vehicle is attached to the ending sequences of the second vehicle. Finally, he does again with the two remaining parts.

The operator keeps the solution which has the best impact on the cost.

Algorithm 7: Pseudo-code of operator cut and reform

```
START
        Select 1 vehicle randomly
        Search for possible cuts
        REPEAT 1 for each other vehicle
                Check for possible cuts
                Check the pair of cut with highest similarity actual time
                Cut the sequences of the two vehicles at the cut find above
                Reform the sequences after switching the two ending sequences
        END REPEAT 1
        Check the cost of each of the solution created above and keep the best
   OUTPUT Best Cut and reform
STOP
```

A schema of an example with two vehicles each having three requests can be seen in Fig 6.

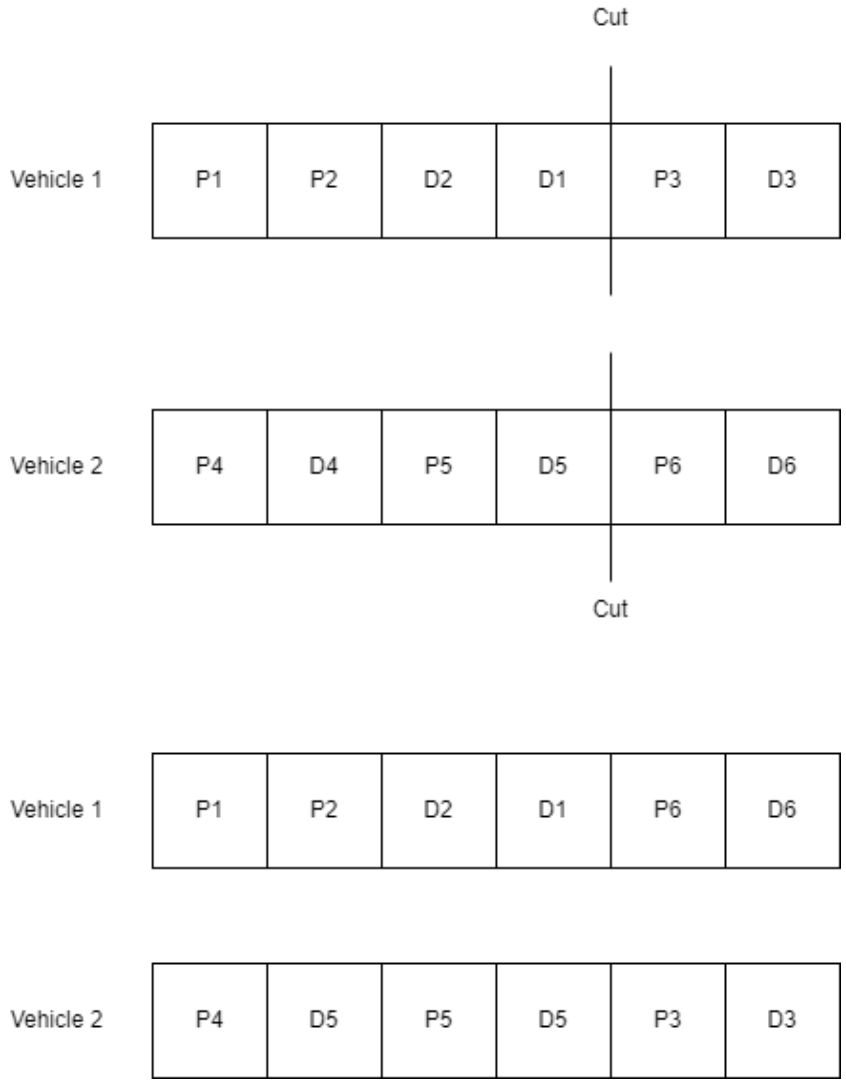

Fig 6: Example cut and reform

This operator is the last one currently used in my algorithm.

Two other local search operators have been designed and drop due to their performance. I will now present them and explain the reasons why they are not included in the algorithm.

f) Swap intra vehicle

This operator was created in order to explore new sequences within a vehicle. This exploration was needed because, during my solution creation process, the nodes are sorted according to their desired times. However, depending on the distances and the maximum ride times, it might cost less to follow another order than simply a chronological one.

In order to do that, a vehicle is selected randomly. For each node of this vehicle, the operator checks if the neighbour node is not the other part of the request[10]. This check is necessary to avoid having a delivery after a pickup. If it is not the case, a swap of the two nodes is done. The swap which has the better impact on the global cost of the solution is kept.

Algorithm 8: Pseudo-code of operator swap intra vehicles

```
START
        Select 1 vehicle randomly
        REPEAT 1 for each node
                Verify if it is feasible to swap this node with either of his neighbour
                 REPEAT 2 for each feasible neighbour
                        Swap the two nodes.
                END REPEAT 2
        END REPEAT 1
        Check the costs of the solutions created above and keep the best
   OUTPUT Best new sequence
STOP
```

Even if theoretically, this local operator seemed indispensable in order to tackle the sorting done is the solution creation process, it turned out to be useless. Over several tests done on various instances of different sizes, ameliorations were hardly ever found after the utilisation of this local operator. His performance was disappointing.

That is why I chose to drop it from the algorithm. It only added computational time without improving the performance of the algorithm.

g) Remove

This last operator goal is to remove a vehicle completely from a solution. Due to the fact that the number of vehicles is a constraint to minimise, it seems reasonable to think that reducing the number of vehicles might improve the overall cost of the solution.

[10] If the node is a pickup, the neighbour can't be his deposit and vice versa

Each vehicle is considered to be removed. This implies that this local operator explores as many new solutions as there are vehicles currently used in the solution. To remove a vehicle, all of the requests of the vehicle are stored in a waiting list. An insertion heuristic is then applied to those requests. The insertion heuristic is the same as the one used in the creation process. An insertion is obviously considered only to vehicles which are already used since the aim is to reduce the number of vehicles. In opposition to the creation process, the order in which the requests are inserted in the other vehicle is not random. Instead, requests are initially ordered by the desired pickup times. Once this is done the solution with the best cost is kept.

Algorithm 9: Pseudo-code of operators remove

```
START
        REPEAT 1 for each vehicle
                Remove all the requests from the vehicle in sort them in chronological order based on
                their desired pickup time in a waiting list
                 REPEAT 2 for each request in the waiting list
                        Insert the request at the best feasible place inside the other vehicles
                END REPEAT 2
        END REPEAT 1
        Check the costs of the solutions created above and keep the best
   OUTPUT Best new solution
STOP
```

This operator was not kept because he is too situational. The solutions that would see their cost decrease thanks to this operator are very rare. Indeed, due to the cost of each additional vehicle, the solution creation process often proposed solution with the minimum amount of vehicle. Furthermore, the operators b and c can reduce the number of vehicles. In addition to that, reducing the number of vehicles will only worsen solutions most of the time it is used during the Deterministic Annealing.

For those reasons, I chose not to use this operator in my algorithm. However, I believe that this algorithm should not be giving up on completely. From a pure managerial point of view, it has a lot of value. For example, a company could consider always keeping a vehicle unused in order to tackle unpredicted events as long as the cost not using this vehicle does not exceed a certain limit. Another possibility is that they aim to use one less vehicle as long as it does not violate any maximum ride time duration to fulfil each request without this vehicle.

Depending on the case, this operator could be added to the algorithm under predefined conditions. For example, if a solution does not have a spare vehicle the operator is used and the new solution is evaluated to see if keeping this spare vehicle is worth in comparison with the cost difference.

4. Results and improvements

This section will be divided into three. In the first one, I will present the computational results obtain with the algorithm as presented it the methodology. Those results will then be used to see the relevancy of possible improvements that I will develop in the second section. Finally, I will present the results of a final version of the algorithm containing the improvements I deemed complete.

The complete results of the various tests performed are available in the Appendix. IV

4.1. Initial results

The results presented here will be the basis to evaluate the possible improvements that will be discussed in the following section.

Those results correspond to the application of my algorithm on five different instances[11]. Each instance is tested five times to minimise the random factor of the algorithm. Indeed, the best solution can be found as soon as the population is created, which could lead to believe the algorithm has performed perfectly in a record among of time. The opposite is also possible. A test could take very long because improving solutions are found each fifth iteration of the algorithm. In those cases, four full iterations of the algorithm are performed for nothing. That leads to a huge time computation for no direct result. In order to avoid the random factor completely, each instance should be tested an enormous amount of time. This is, however, not possible if I want to test several improvements as each test takes times.

Among those instances, two are considered small (28[12] and 46[13] requests), two mediums (53[14] and 56[15] requests) and one large (90[16] requests). The number of instances is purposefully quite low. Once again this is done to allow testing more improvements. Indeed, each improvement must be tested several times on several instances.

In order to simplify the comprehension of the results, the instances will be referred by their sizes such as I28 for the instance with 28 requests. Those instances were chosen among all the one proposed by Chassaing et al. (s.d) for no specific reason other than their sizes.

Results will be presented into two sections. The first will focus on the balance between the quality of the solution, and the computational time needed to find them. This section will be used to evaluate possible improvement to the performance of the algorithm. Those results will be presented through the best solutions found (Best), the average solution found (Avg), the average deviation between the

[11] The modified instances used can be found at
https://drive.google.com/drive/folders/1wx_AEqRCrkUJOiPrXUOpoJ27avGGXNbZ?usp=sharing
[12] Instance RL_d55
[13] Instance RL_d01
[14] Instance RL_d96
[15] Instance RL_d30
[16] Instance RL_d02

first two (Dev) in percent and finally the CPU presented in minutes. The average deviation allows monitoring the random factor of the algorithm. The smaller this deviation is, the more consistent the algorithm is.

The second will focus on the distribution of the total cost among his different components. The components being the cost of the distances, the cost of the number of vehicles used and finally the cost of the delays. This section will be used to see the impact of the addition of new constraints to the distribution between those three costs. For that, I will present the average percentage of each cost. The actual cost value will not be presented in this part as they are not interesting to see the impact of new constraints. Indeed, it is reasonable to assume that adding constraints will modify the global cost, but it is not what we are interested in. Indeed, adding constraints is done in order to correspond more to reality. On the other hand, the impact of a constraint on the cost distribution is interesting as it shows how getting closer to reality impact the DARP. In addition to that, I will monitor the utilisation of the fleet of vehicles.

Finally, I will monitor the number of times the large search operator is used successfully at the end of the hybrid BA with DA. This is a good indicator of the overall precision of the algorithm. Indeed, if the LNS find a better answer, it is an answer that the hybrid could have found. This reasoning, of course, has his limits as the fact that the LNS find a solution also depend on randomness. A really bad solution could have no better one in his neighbourhood meaning he is a local minimum. On the other hand, a really good solution could have one simply due to the fact that the operator used here is different from the other. Those cases are, however, less likely. That is why it remains a good indicator of the overall precision.

4.1.1. Performance of the algorithm

	Best	Avg	Dev (in %)	CPU (In min)
I28	3037	3301.2	8	5.92
I46	6709	6945.2	3.4	9.47
I53	7404	7838.4	5.54	10.39
I56	5545	6170.2	10.13	10.67
I90	10273	10852.6	5.34	19.19

Table 3: Initial algorithm's performance

The deviations are quite high, and the best solutions are never found more than once. This is due to the fact that I have the number of iterations of the Deterministic Algorithm allowed is low. My algorithm only does around 150 iterations of the DA each time it is used. If this algorithm was to be used in real situations, the stopping criterion must be adapted. However, for the purpose of this thesis, this stopping criterion remains interesting as it allows testing a lot of different scenarios, constraints or parameters.

4.1.2. Cost distribution

	Cost of distances	Cost of delays	Cost of vehicle utilisation
I28	59%	12%	29%
I46	60%	13%	27%
I53	58%	11%	31%
I56	59%	12%	29%
I90	57%	11%	32%

Table 4: Initial algorithm costs' distribution

The distribution between the three costs is influenced by a variety of factors such as the area of service, the fleet available, the cost attributed per vehicle and so on. It is, however, interesting to see that the cost distributions are similar from instances to instances.

	Fleet of vehicles	Number of vehicles used in my algorithm	Number of vehicles used by Chassaing et al. (s.d)
I28	5	5	4
I46	7	6	7
I53	11	8	10
I56	8	8	8
I90	13	12	13

Table 5: Utilisation of vehicles in the best solution for algorithm initial

Except for the smallest instance, my algorithm tends to use fewer vehicles than Chassaing et al. (s.d). Unfortunately, the objective function, constraints and method they used to find those results are not specified. However, hypotheses can be made to explain those differences. For example, it is possible that the number of vehicles used is not a value to minimise in their formulation. Another justification could be that they minimise the number of vehicles but the cost per vehicle they used in smaller than mine.

4.1.3. Performance of the LNS

Improvement was found using the large search operator in 68% of the tests. As the size of the instance rise, this number of times the LNS is used is higher. This is due to the fact that the LNS find solutions more frequently if the solution is not already precise. The algorithm cannot explore as much as it should during the DA for the large instance. This is due to the fact that they take longer to do an iteration but have the same amount of time as the other instance. This is the drawback of having an amount of time for stopping criterion instead of a number of iterations. However, this choice was made in order to save time which it does.

4.2.Improvements

An algorithm is never completely finished. There is always a possibility to improve it. For that, there are two types of improvements.

The first are improvements to the performance of the heuristic. In order to do that, it is not enough to improve only the precision or the computational time of the algorithm alone if it deteriorates the other. Those two elements must be considered together. A heuristic can find better solutions given more time but it is not the direct purpose of the heuristic. It should find good solutions in acceptable amount of time. That is why heuristic are evaluated according to their balance between precision and computational time. An improvement is considered beneficial if it improves this balance.

The second type is the realistic aspect of the algorithm. DARP are modelled in order to solver real life situations. However, life is unpredictable. Unexpected events are doomed to happen. The job of the algorithm is to give a good solution to guide a decision maker but it cannot predict the future. It is, however, possible to take into account real live constraints in order to reduce the effect of unexpected events. Another balance must be taken into account here. It is always possible to add real life constraints to the problem. However, each additional constraint risks to augments the computational time needed to solve the algorithm. At the end, the algorithm remains a way of solving a modelling of reality which could never be perfect. That is why one must decide if it is worth trying to predict unexpected events at the cost of computational time.

Improvement of the first kind includes improving the structure, the local operators, the parameters and so on. Those can be easily tested through the balance between time and computational cost. In this section, two improvements of this type will be tested: improving the parameters and the LNS.

On the other hand, improvements of the second type stay at the appreciation of the situation in which they are implemented. One must decide if he deemed this improvement worth or not. I will explore two possible improvements of this kind: adding a maximum ride time for the vehicles and having a time dependent speed.

4.2.1. Parameters tuning

One of the main methods to improve the performance of an algorithm is to improve his parameters. This is called parameters tuning. For that, a search for better parameters is conducted.

The search for the best parameters is an endless task as some parameters might be perfect for a certain set of instances and not for others. Furthermore, as the number of parameters augment, the combinations of them rise exponentially. Knowing that for each parameter there are an infinite number of possibilities, finding the best combination of a group of parameters is a near-impossible task.

The task is then to find a good combination of parameter fitting acceptably any set of instances. For that, one must use recommendations of literature, follows his intuition and finally test several

possibilities. In addition to that, Hyperparameter Optimisation Tools have been developed in order to explore various combinations of parameters.

Due to the difficulty of tuning parameters, I will only focus on the parameters of the Deterministic Algorithm. I have made this decision for two reasons.

The first reason is that, as we have seen in the section 2.3.2.10, the parameters of the Bee algorithm do not have a big influence on the global performance of the algorithm. In addition to that, they have been chosen to minimise the computational time. For the parameter of the tournament selection, he has already been explored during the implementation of the initial algorithm.

The second reason is the stopping criterion I used in my Deterministic Algorithm. As much as it is useful in order to test every size of instances while keeping reasonable computational times, it makes the tuning of the parameter harder. Indeed, several of the parameters of the Deterministic Annealing are directly related to the number of iterations performed. This is the case of the threshold reduction ΔT and the reset parameter n_{imp}. That is why finding good parameters which perform well on instances of various sizes is critical.

As we have seen in the section 2.3.2.10, Deterministic Annealing has three parameters $T_{max}, \Delta T$ and n_{imp}.

For T_{max}, I will keep the same logic which is to set T_{max} to the average distance between two locations of the instances multiplied by a new parameter t_{max}. However, I will try to find a better value for t_{max}.

For ΔT, I will also keep the same logic but I will go a bit further. I decided to set ΔT to be a proportion of T_{max}. Following my intuition, I chose to take 1/75 of T_{max}. Here, instead of basing the setting to my intuition, I will define a new parameter called Δt. This parameter is the number by which T_{max} is divided. This gives $\Delta T = \frac{T_{max}}{\Delta t}$. The next step is to find the best value for Δt.

Finally, for n_{imp}, I will not use five times the number of vehicles currently used. I will explore n_{imp} independently.

This gives me three parameters to tune: t_{max}, Δt and n_{imp}. Even though, there are used inside the hybrid BA with DA, I will explore them as the DA was used out of the hybrid. This will reduce considerably the time needed to explore a combination. Furthermore, the hybrid BA with DA does not influence the performance of the DA. It only determines when and how many times it is used. It has no influence on the parameters.

In order to do that, I will use Hyperparameter Optimisation available on Julia called Hyperopt. This tool explores several combinations of parameters on a function and returns the best combination. However, to the extent of my knowledge, it does not allow testing several sets of data. I was only able to test one instance at a time. For that reason, I did a Hyperparameter Optimisation on three different

instances; a small one (34[17] requests), a medium one (55[18] requests) and a large one (76[19] requests). Those instances are not the same as the ones as I usually use for the computational results. This has been done in order to avoid having overfitting parameters. Overfitting being a phenomenon where parameters work perfectly for some instances but yield poor results if tested over a large group of different instances. I don't want parameters that will be perfect for my five testing instances. This is not the goal of this parameter tuning. I want to find parameters that will work well for any instance.

In order to work, Hyperopt needs a function to test the parameters, a range and spacing size for each parameter and the number of tests he will perform. The function is obviously the Deterministic Annealing but the other components disserve more explanations.

For t_{max}, I followed the tests done by Braekers et al. (2014) which tests value for between 0.3 and 3. The size of the spacing determines how many values in the defined range can be selected to be tested. For example, for a range from 1 to 5, if we choose the spacing of 5, the possible values are 1,2,3,4 and 5. In the case of t_{max} I chose the spacing of 10.

For Δt, I selected the range according to tests I made to identify the number of iterations done by the DA before the stopping criterion is reached. I set it to be 50 to 250 with the spacing of 200.

Finally, for n_{imp}, I only followed my intuition and set it to have a range of 10 to 50 with the spacing of 40.

Since the Hyperparameter Optimisation will be conducted three times, I cannot allow Hyperopt to conduct too many tests. That is why I decided to set the number of tests to be performed to 200. Each test tries a combination of parameters. This led to a run time of Hyperopt of 1.5 hours. This number of tests obviously only explores a minuscule fraction of possible combinations since, with those ranges, there are 80000[20] possible combinations. However, exploring all those combinations would take 15 days for each instance.

Hyperopt found me the following combinations of parameters for the three instances.

	t_{max}	Δt	n_{imp}
Small instance (I34)	0.9	150	25
Medium instance (I55)	3	95	34
Large instance (I76)	2.1	86	23

Table 6: Parameter found with Hyperopt

[17] Instance RL_d10
[18] Instance RL_d47
[19] Instance RL_d76
[20] $10 \times 200 \times 40 = 80000$

To facilitate further explanation, the parameter of the small instance will be called Set 1, those of the medium instance Set 2 and finally, those of the large instances Set 3.

Now that I have three combinations of parameters, I need to see which one works better among different sizes of instances. In order to do that, I will test them not on the same instances used to find them but rather on the instances used to test the performance of the algorithm.

The new parameters will be tested three times on a small, a medium and a large instance. The results will then be compared to identify which combination of parameters seems to be the best. The result for the different sets can be seen in the tables, 7, 8 and 9.

	Best	Avg	Dev (in %)	CPU (In min)
I46	6692	6722,33333	0.45	10.6
I56	5964	6023	0.97	9.7
I90	10577	10696	1.11	8.96

Table 7: Performance with Set 1

	Best	Avg	Dev (in %)	CPU (In min)
I46	6409	6552,66667	2.19	11.16
I56	6120	6276,66667	2.49	8.2
I90	10193	10455	2.5	16.95

Table 8: Performance with Set 2

	Best	Avg	Dev (in %)	CPU (In min)
I46	6570	6713,66667	2.13	7.92
I56	5738	5899,33333	2.73	15.67
I90	10180	10265	0.82	20.71

Table 9: Performance with Set 3

Despite the fact that the first set of parameters gives the results with least deviation, the general quality of the solutions is worse than with the other two set. Even their general low computational time can't justify choosing this set.

The choice between the set two and three is more difficult. The Set 3 gives better results but at the cost of a higher computational time. However, several compromises have already been done to reduce the computational time. That is why, I chose to keep the Set 3 as the best improving parameter combination among the three sets.

Another comparison must be made before accepting those new parameters. The performance of those parameters must be confronted with the performance of the parameters of the initial algorithm. For

that, I have conducted two more tests on each of the three instances above and five tests over the other two instances of the comparison pool. In the Table 10, the first 6 lines correspond to the initial result and the last 5 the results with the new parameters.

	Instance	Best	Avg	Dev (in %)	CPU (In min)
Algorithm initial	I28	3037	3301.2	8	5.92
	I46	6709	6945.2	3.4	9.47
	I53	7404	7838.4	5.54	10.39
	I56	5545	6170.2	10.13	10.67
	I90	10273	10852.6	5.34	19.19
Algorithm with new parameters	I28	3118	3208,2	2.81	6.27
	I46	6492	6633,4	2.13	8.35
	I53	7269	7607,4	4.44	9.29
	I56	5738	5911,2	2.93	13.91
	I90	10180	10320	1.35	18.03

Table 10: Performance comparison between initial parameters and new ones

There is no debate that the new parameters are better than the one I used initially. The average is improved for every instance. The deviation is smaller which implies less randomness while testing. Furthermore, the computational time is smaller for 3 of the 5 instances.

However, it is worth mentioning that the best solutions found for the instances 28 and 56 have been found with the initial parameter. This is due to the randomness of the algorithm. If more test is conducted, that might not be the case anymore.

Another indicator of the performance of the algorithm is the fact that the LNS has only been used in 44% of the test instead of the previous 68%.

The new parameters should be used instead of the initial ones as they are a clear improvement of the balance between precision and computational time. They might, however, not be the best parameters possible but they are reasonably good.

4.2.2. Larger operator for Large Neighbourhood Search

Even if the operator relocate was too wide to be used inside the DA, it is, however, too small to be really considered an operator for LNS. That is why I have developed another operator derived from the operator relocate. I have called him expanded relocation.

What it does is applying a relocation on every request. Then in each of the solutions found thanks to those relocations, we apply the current operator relocation. It explores solutions two relocations away. A pseudo-code of this operator can be seen in the Algorithm 10.

In order to do that, the current operator relocation is used twice. It is used a first time but return all the solutions created by relocating requests. Then for every of those solutions, we apply again the

operator relocate. If the second relocation improves the cost of the solution, the operator keeps this new solution. Otherwise, the operator keeps the solution prior to the second relocation.

Algorithm 10: Pseudo-code of operator expanded relocate

```
START
        Apply operator relocate with all the solutions as OUTPUT
        REPEAT for each solution
                Apply operator relocate to the solution
                KEEP the best of the solution after first relocation or the second one
        END REPEAT
        Check the costs of the solutions created above and keep the best
   OUTPUT Best extended relocation
STOP
```

In order to test this possible improvement, I will not only monitor the usual factor but also the time needed to perform the LNS. Indeed, the expanding the LNS could lead to finding a better solution, but it must remain an extra step after the hybrid BA with DA. Otherwise, it becomes a new algorithm completely.

The test on the smallest instance went well with time under 2 seconds to perform the LNS. However, as I tested larger instances the time needed to perform it rises considerably. It took 30 seconds for the instance of size 46 but rose already to 161 for the one of size 53. It was already a third of the total time of the algorithm. I tested it also on the larger instance and the time needed to perform the LNS rose to 2112 for a total time of 2800. This is obviously not ideal as this LNS must remain an extra step to search further and not the main part of the algorithm.

Those results can be explained through 2 factors.

The first one is that the operator takes longer for larger instances as he must explore more relocation. This one was expected as all operators follow this logic.

The second one, on the other hand, was not. The overall precision of the algorithm drops as the instance grow larger. This is due to the fact that the stopping criterion of the DA is based on an amount of time. For small instances, the DA can perform considerably more iterations than on a larger one. This leads to a drop of precision for the larger one. If the hybrid is less precise, the solution on which the LNS is applied is less precise. This leads to more solutions found by the LNS. All of that gives us a LNS that takes longer for large instances and is used for more iterations.

For those reasons, this larger operator is not fitted for the current algorithm. Nevertheless, if we improve the precision of the hybrid DA with SA by allowing more time for the DA, this operator can be kept as a last search for possible improvement. The time needed to perform the LNS would remain the same per iteration but fewer iterations would be needed as the algorithm has already found a good solution. Furthermore, the computational time of the rest of the algorithm would be higher thus the time needed to perform the LNS would seem more acceptable.

4.2.3. Maximum ride time for vehicle

In real life, maximum ride time for a vehicle is mandatory to respect drivers working hours. That is why I chose to add this constraint to the algorithm.

For the same reason as I have allowed violation of the maximum rides times of customers to diversify my population, I will do the same for the maximum ride's times of the vehicles. However, the violation fee must be much higher as it is more expensive to pay for a driver overtime rather than letting some customers wait longer. I chose a violation fee 10 times the violations for customers.

Several tests were conducted and it appeared that it is nearly impossible to find solutions which respect both customers ride times and vehicles ride times. In order to tackle that issue, there are several possibilities.

The first is to organise two shifts. The requests have desired times varying between 4h30 and 21h. This gives us a time window of 16.5 hours. With this in mind, it is possible to have a shift that goes from 4h30 to 13h and the next to 13h to 21h. However, this would mean that the company would need twice the staff. From the algorithm perspective, that means forcing a return to the depot around 13h which will correspond to the shifts change. The cost per vehicle could be transformed into cost per shift in order to minimise the staff.

The second would be to refuse some requests in order to respect the maximum ride times of the vehicles and the other requests. For the algorithm, a waiting list could be created. This waiting list will contain the requests that are not included in the current solution. Furthermore, an operator can be created in order to withdraw a request from the solution to replace it with one from the waiting list. Another possibility is to take into account the benefit a company has to serve a customer. Indeed, longer distance might bring more benefit than short one due to the pricing system. With that in mind, it is more efficient to serve customers who will make the company gains more money. The other will be sent to the waiting list and ultimately refused.

Finally, it is also possible to increase the number of vehicles or simply to allow maximum ride time violation for customers even if it decreases the service quality.

All of those possibilities depend of managerial decision which will be different from a situation to the next. That is why, the addition of this constraint will not be further explored.

4.2.4. Time-dependent speed

The last possible improvement that will be discussed in this thesis is a time-dependent speed. The idea behind it is to take into account the traffic in the algorithm. Indeed, a constant speed of 60 km/h like supposed in the algorithm is an oversimplification. In practice, the time needed to travel from a point to another depends on a variety of factors such as the moment of the day, the type of road use and so on.

Here we will focus on the evolution of the average speed depending on the time of the day. The traffic evolves during the day. Very early in the morning, the traffic will be very fluid as people are still asleep. Then between 7 and 9h, it will be considerably slower as it is the time of the day where people go to work. After that and until 16h, the traffic is fluid again. Between 16 and 19h, we have the same effect as between 7 and 9h as people return home from work. The traffic is calm again after.

This gives us two-time windows during which the traffic is slower. This leads to the average speed of vehicle travelling within those time windows being smaller than during the rest of the day. However, representing perfectly the traffic is impossible. To implement time dependent speed in this algorithm, I will assume five time windows: [4h-7h], [7h-9h], [9h-16h] [16h-19h] [19h-21h]. Furthermore, I will work with two speeds. For peak traffic time windows, I will assume a speed of 40 km/h and for the rest 90 km/h.

In order to implement the dependent speed into the algorithm, it is not enough to check the time window corresponding to the departure time to set the speed accordingly. Indeed, travel might begin within a time window and end in another. In this case, the vehicle is assumed to travel at the speed of the first time window until he reaches the end of it then switches to the speed of the next. This is, of course, not how it would happen in reality but modelling has its limits.

To implement this concept, I have used the model described by Ichoua et al. (2001). The procedure they propose is the following:

1. set t to t_0,
   set d to d_{ij},
   set t' to $t + (d/v_{cT_k})$.
2. while $(t' > \bar{t}_k)$ do
   2.1 $d \leftarrow d - v_{cT_k}(\bar{t}_k - t)$,
   2.2 $t \leftarrow \bar{t}_k$,
   2.3 $t' \leftarrow t + (d/v_{cT_{k+1}})$,
   2.4 $k \leftarrow k + 1$.
3. return $(t' - t_0)$.

Fig 4 – Travel time calculation procedure. Reproduce from "Vehicle dispatching with time-dependent travel times", by Ichoua, S., Gendreau, M., Potvin, J-Y. (2003). Vehicle Dispatching with time – dependent travel times. European Journal of Operational Research. 144. 379-396. 10.1016/S0377-2217 (02)00147-9.

With t_0 being the departure time, d_{ij} the distance between the two points, t_k the limit of the time windows before changing the speed and v_{cT_k} the speed of the time windows k. The c is not used in my algorithm but allow specifying specific speed for a node type c.

What the procedure does is checking if the initial arrival time t' calculated in the setting passed through a time window limit. Each time it does the travel time is recalculated to fit the speed of this time window.

This procedure has been added to the algorithm and tests were conducted three times on the same 5 instances. Even though, this is considered to be an additional constraint, I will also present the total cost. The reason behind that is that it is not guarantee that the costs will rise because the speed of some time windows is bigger than the initial average one.

	Instance	Best	Avg	Cost of distances	Cost of delays	Cost of vehicle utilisation
Algorithm initial	I28	3037	3301.2	59%	12%	29%
	I46	6709	6945.2	60%	13%	27%
	I53	7404	7838.4	58%	11%	31%
	I56	5545	6170.2	59%	12%	29%
	I90	10273	10852.6	57%	11%	32%
Algorithm with time-dependent speed	I28	3002	3047,33333	59%	10%	32%
	I46	6187	6313,33333	61%	7%	31%
	I53	7272	7542,33333	59%	6%	35%
	I56	6035	6428	63%	14%	23%
	I90	10698	10941,6667	59%	9%	32%

Table 11: Performance comparison with and without time-dependent speed

The version with time-dependent speed tends to use more vehicles than the initial one. On the other hand, the costs are in general not higher. This is due to the fact that the time windows with traffic represent a smaller portion of the service time than the rest. This leads to vehicles generally driving faster than with the average speed of 60 km/h. It also brings the cost of delays down. This is due to the fact that as a vehicle drive faster, it reduces travel times. However, this reduction is probably not enough to allow the insertion of new nodes between two points. This leads to sequences remaining the same but with fewer delays.

It would be possible to go even further in this logic. The speed could be time and zone-dependent. The zone could represent the urbanisation level of the area. Indeed, highly urbanised areas tend to have more traffic than rural ones. Each node would have an additional data based on their urbanisation level. In opposition with the processing of time, it is unknown when a vehicle will leave this zone. For this factor, it must only depend on the zones of the departure and destination points.

However, in order to do that a good knowledge of the locations is needed. That is why it will not be explored further here.

4.3.Final version

This final version will include both the time-dependent speed and the new parameters as those improvements have a good effect on either the performance of the algorithm for the first or the realistic aspect for the second.

The best solution for each instance is available in the Appendix III

	Best	Avg	Dev (in %)	CPU (In min)
I28	2841	2959,6	4	6,02666667
I46	6292	6418,2	1.96	12,03
I53	7204	7378,8	2.36	12,4466667
I56	5653	5766	1.95	17,86
I90	9575	10216	6.2	22,18

Table 12: Final algorithm's performance

Since both the new parameters and the time-dependent speed was reducing the costs of the solutions, the results above have the best average of all the tests I conducted. Furthermore, four of the best overall solutions were found was found with this version. Only for the instance 56, a better solution was found with the initial algorithm. However, the speed was not the same.

	Cost of distances	Cost of delays	Cost of vehicle utilisation
I28	60%	7%	33%
I46	61%	7%	32%
I53	57%	7%	36%
I56	62%	10%	28%
I90	57%	7%	36%

Table 13: Final algorithm costs' distribution

The cost distribution remains similar except for the instance 56.

	Fleet of vehicles	Number of vehicles used in my algorithm	Number of vehicles used by Chassaing et al. (s.d)
I28	5	5	4
I46	7	7	7
I53	11	9	10
I56	8	7	8
I90	13	12	13

Table 14: Utilisation of vehicles in the best solution for algorithm final

The number of vehicles changes due to the effect of the time-dependent speed. They remain overall smaller than the one of Chassaing et al. (s.d).

The instance 56 has the most out of the norm results. Their cost was not smaller, their cost distribution not similar and they use fewer vehicles. The last part is particularly interesting because in all the tests I have conducted, the instance 56 had the less variance in the number of vehicles used from solutions to solutions. The solutions used constantly 8 vehicles. This might due to the fact that this instance benefits a lot of the time-dependent speed. Without it, it was not possible to find feasible solutions without using all the vehicle. The benefice might not seem obvious due to the fact that the best solution was not found here but this is probably due to randomness.

The LNS was used in 40% of the case. This is very similar to the test with the new parameters alone. This is expected as even if the time-dependent speed reduced the cost, they have no influence on the performance of the algorithm.

5. Conclusions

Dial a ride problem is used to represent several real-life door-to-door transportation. The clearer example being the transportation of the elderly or disabled people. With the aging of the population, the demand for those services will rise. That is why good optimisation tools are needed in order to help companies or states that want to offer those services.

However, in opposition with most transportation problems, DARP must focus not only on the cost minimisation but also the human factor. Indeed, people that require those kinds of services have a lot of requirements. That can go from maximum ride time for health reasons to specific accommodations. A balance between minimisation cost and inconvenience for the user must be found.

In this paper, I have presented a possible algorithm to solve Dial a ride problem. In order to do that, I had to choose which constraints I would tackle and which cost I would minimise. I made those choices based on my intuition and recommendation from literature but they must be adapted to fit the reality of the situation one must solve. Furthermore, it exists several possible algorithms. I have chosen to present a hybrid Bee Algorithm with Deterministic Annealing as it was proven to yield good results but there is no perfect algorithm that would be the best for all the different situations. Several other options have been presented in the literature review and reader are referred to various papers depending of the algorithm of their choice.

In order to use my algorithm in a real-life situation, the first step once must make to use this algorithm is to augment the stopping criterion of the DA. It was purposefully tuned down in order to allow fast testing. That is why I would recommend keeping those tuned down parameters to adapt the algorithm to the situation one must solve. After that, the stopping criterion can be increased. Once this is done, one must tune the parameters accordingly following either the method I used a similar one such as irace[21] which is not specific to Julia. The constraints and objective function must also be adapted to the specific situation.

Several situations have been discussed in this paper. To cite only a few: using an operator removes to leave a vehicle available, areas dependent speed, sequence with two shifts corresponding to two drivers and so on. The reality is such that it is always required to adapt the algorithm as new factors presents themselves. Real-life situations are not usually described in literature. That is why more research could be conducted with this logic to answer real-life problems.

[21] For more information: https://doi.org/10.1016/j.orp.2016.09.002

The version presented here remains unfinished as it is always possible to improve it. Some improvements were explored in this paper but it is always possible to go further. Ideas that were not explored in this paper includes multiple depots, heterogenous vehicles, operators with roulette selection or lunch and coffee break for drivers. The biggest improvement remains the utilisation of strict time constrains like time windows instead of relaxed such as desired times. Several of those improvements have been developed in literature in paper such as Masmoudi et al. (2016) or Braekers et al. (2014). Finally, operators being the key part of most algorithms, it is always possible to explore new operators in order to keep the operators that yield the best results.

To conclude, dial a ride problem is a wide topic as each situation is different. It means that it is always possible to search further. However, the goal must remain to help companies and states that are searching to provide those services. Studies must be conducted according to their needs.

6. List of Figures

Figure 1: Empty example of dial a ride problem

Figure 2: Completed example of dial a ride problem

Figure 3: Flowchart of the hybrid Bee Algorithm with Deterministic Annealing

Figure 4: Solution example

Figure 5: Example solution representation

Figure 6: Example cut and reform

7. List of Tables

Table 1: Summary of literature review

Table 2: Algorithm comparison

Table 3: Initial algorithm's performance

Table 4: Initial algorithm costs' distribution

Table 5: Utilisation of vehicles in the best solution for algorithm initial

Table 6: Parameter found with Hyperopt

Table 7: Performance with Set 1

Table 8: Performance with Set 2

Table 9: Performance with Set 3

Table 10: Performance comparison between initial parameters and new ones

Table 11: Performance comparison with and without time-dependent speed

Table 12: Final algorithm's performance

Table 13: Final algorithm costs' distribution

Table 14: Utilisation of vehicles in the best solution for algorithm final

8. List of Algorithms

Algorithm 1: Pseudo-code of the proposed hybrid Bee Algorithm with Deterministic Annealing

Algorithm 2: Pseudo-code of the proposed Deterministic Annealing

Algorithm 3: Pseudo-code of operator relocate

Algorithm 4: Pseudo-code of operator relocate 1 to 1

Algorithm 5: Pseudo-code of operator relocate 2 to any

Algorithm 6: Pseudo-code of operator swap

Algorithm 7: Pseudo-code of operator cut and reform

Algorithm 8: Pseudo-code of operator swap intra vehicles

Algorithm 9: Pseudo-code of operator remove

Algorithm 10: Pseudo-code of operator expanded relocate

9. Appendices

I. Summary of several algorithms for the dynamic multi-vehicle DARP

Reference	Objective	Time windows	Other constraints	Algorithm	Size of instances solved
Madsen et al. (1995)	Multi-criteria objective	On pickup or delivery	Several vehicle types. Vehicle capacity. Maximum route duration. Maximum deviation between actual and shortest possible rid	Heuristic. Vertex insertions	$n = 300$
Teodorovic and Radivojevic (2000)	Minimise a function incorporating route lengths, ride times and time window violations	On pickup and delivery	Vehicle capacity	Sequential insertion of users in vehicle routes. Nine rules are used to give more or less weight to the various elements of the objective	$n = 900$
Colorni and Righini (2001)	Maximise the number of serviced requests or maximise the perceived level of service, or minimise the total travelled distance	On pickup and delivery	Vehicle capacity. Maximum route duration	Alternation between clustering and routing algorithms. Branch-and-bound algorithm is applied to sequence a subset of users with time windows not too far in the future	None
Coslovich et al. (2006)	Minimise user dissatisfaction	On pickup and delivery	Deviation from desired	Insertions in current	$25 \leq n \leq 50$

			service time. Upper bound on 'excess ride time'	routes. Route reoptimizations with modified 2-opt	

Table 3 – I. Summary of several algorithms for the dynamic multi-vehicle DARP. Reproduce from 'The dial-a-ride problem: models and algorithms', by Cordeau, J.-F., Laporte, G., *Ann. Oper. Res. 153 (1),* 29–46.

II. Identification for the best parameters setting for the hybrid BA-DA (BA-SA).

Table 3
Identification for the best parameters setting for the hybrid BA-DA (BA-SA).

N		10	20				30			
(*be*, *es*)		(5,3)	(15,10)	(15,5)	(10,5)	(10,3)	(20,10)	(20,5)	(15,10)	(15,5)
BA-SA	Best	813.15	**811.88**	**811.88**	**811.88**	812.05	**811.88**	**811.88**	**811.88**	**811.88**
	Avg	815.63	813.96	**814.23**	815.06	814.88	813.21	814.64	814.51	815.34
	CPU	18.75	30.05	**25.76**	27.98	32.75	35.87	31.42	39.53	43.86
BA-DA	Best	812.75	812.03	**811.88**	**811.88**	**811.88**	**811.88**	**811.88**	**811.88**	**811.88**
	Avg	813.12	814.23	**815.32**	815.75	814.63	813.29	814.32	815.65	815.82
	CPU	17.96	27.23	**25.65**	26.53	31.02	30.86	35.29	43.23	38.12

Table 3. Identification for the best parameters setting for the hybrid BA-DA (BA-SA). Reproduce from 'Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem', by Masmoudi, MA. Hosny, M. Braekers, K. Dammak, A. *Transportation Research: Part E*. 2016; 96:60-80.

III. Best solutions with final algorithm

Instance 28

[19 330 330 0 4; 7 360 406 46 8; 35 421 421 0 4; 47 428 440 12 0; 8 600 600 0 3; 36 663 663 0 0; 20 840 840 0 1; 21 870 870 0 2; 48 969 969 0 1; 49 975 997 22 0]

[23 390 390 0 3; 51 444 444 0 0; 11 570 570 0 2; 39 626 626 0 0; 1 1080 1080 0 4; 29 1103 1103 0 0]

[22 330 330 0 1; 24 360 360 0 4; 18 360 361 1 5; 28 390 390 0 6; 50 397 397 0 5; 46 415 415 0 4; 25 420 420 0 8; 52 420 423 3 5; 53 489 489 0 1; 56 491 521 30 0; 4 630 630 0 4; 2 660 660 0 5; 32 672 675 3 1; 30 800 800 0 0; 10 840 862 22 2; 38 947 947 0 0]

[3 360 360 0 3; 9 360 422 62 7; 31 411 411 0 4; 37 439 439 0 0; 12 570 570 0 4; 40 619 619 0 0; 15 810 810 0 1; 43 851 851 0 0]

[13 360 360 0 3; 5 360 361 1 4; 17 360 362 2 5; 14 360 365 5 8; 42 409 409 0 5; 33 416 416 0 4; 41 456 456 0 1; 45 464 464 0 0; 26 600 600 0 2; 54 658 658 0 0; 6 660 736 76 3; 34 758 763 5 0; 27 840 840 0 4; 16 870 870 0 8; 44 932 932 0 4; 55 985 985 0 0]

Instance 46

[45 360 360 0 1; 27 360 425 65 5; 25 360 400 40 6; 20 390 390 0 8; 71 433 433 0 7; 66 462 495 33 5; 91 464 464 0 4; 73 480 512 32 0; 34 840 840 0 1; 80 933 933 0 0]

[22 300 300 0 2; 41 360 360 0 3; 68 362 362 0 1; 5 390 390 0 2; 40 420 420 0 6; 87 439 439 0 5; 51 453 453 0 4; 86 484 484 0 0; 3 600 600 0 1; 49 653 653 0 0; 28 810 810 0 2; 38 840 848 8 5; 74 869 869 0 3; 84 926 926 0 0; 12 930 937 7 1; 58 951 951 0 0; 39 1080 1080 0 1; 85 1177 1177 0 0]

[7 360 360 0 3; 8 360 362 2 5; 33 390 390 0 7; 53 431 431 0 4; 54 431 455 24 2; 79 518 518 0 0; 13 600 600 0 3; 32 630 630 0 4; 78 683 683 0 3; 59 774 774 0 0; 30 840 840 0 2; 21 870 870 0 4; 76 946 946 0 2; 67 972 974 2 0; 4 1080 1080 0 1; 50 1179 1179 0 0]

[17 330 330 0 1; 6 360 360 0 2; 43 390 390 0 4; 46 390 394 4 8; 63 412 412 0 7; 52 439 439 0 6; 89 458 458 0 4; 92 547 547 0 0; 18 600 600 0 1; 19 630 630 0 2; 64 661 661 0 1; 65 682 682 0 0; 24 810 810 0 1; 70 912 912 0 0]

[9 330 330 0 1; 16 360 360 0 2; 42 360 380 20 3; 11 360 399 39 6; 55 398 398 0 5; 62 434 434 0 4; 88 439 457 18 3; 57 461 516 55 0; 2 600 600 0 1; 48 646 646 0 0; 31 780 780 0 1; 23 810 904 94 2; 77 879 890 11 1; 69 926 926 0 0; 26 1080 1080 0 3; 35 1110 1114 4 4; 72 1134 1134 0 1; 81 1139 1141 2 0]

[; 37 330 330 0 1; 15 360 360 0 5; 29 360 361 1 6; 36 390 390 0 7; 83 406 406 0 6; 61 460 460 0 2; 75 532 532 0 1; 82 534 534 0 0; 44 570 570 0 2; 1 600 608 8 5; 90 637 637 0 3; 47 695 695 0 0; 10 840 840 0 4; 56 910 910 0 0; 14 1050 1050 0 1; 60 1111 1111 0 0]

[][22]

[22] Empty vehicle

Instance 53

[45 360 360 0 1; 15 390 390 0 2; 98 442 442 0 1; 68 494 494 0 0; 12 600 600 0 1; 20 600 601 1 2; 73 677 677 0 1; 65 714 714 0 0; 36 840 840 0 1; 14 870 870 0 3; 89 935 935 0 2; 67 971 971 0 0; 19 1050 1050 0 4; 72 1153 1153 0 0]

[32 270 270 0 1; 85 343 343 0 0; 24 360 381 21 4; 77 398 400 2 0; 25 420 465 45 2; 78 549 549 0 0; 38 600 600 0 3; 18 600 603 3 6; 91 666 666 0 3; 71 696 696 0 0; 47 840 840 0 2; 43 870 870 0 5; 100 919 920 1 3; 96 973 973 0 0]

[16 330 330 0 2; 11 360 360 0 5; 48 360 361 1 6; 101 415 415 0 5; 69 444 444 0 3; 64 599 599 0 0; 39 660 660 0 1; 92 769 769 0 0; 7 840 840 0 1; 17 840 844 4 5; 60 940 940 0 4; 70 940 944 4 0; 23 1080 1080 0 1; 76 1128 1128 0 0]

[5 330 330 0 2; 31 360 380 20 4; 58 405 413 8 2; 84 436 461 25 0; 22 600 600 0 1; 75 742 742 0 0; 50 810 810 0 1; 103 837 837 0 0; 29 840 868 28 1; 82 887 889 2 0]

[51 360 360 0 1; 104 429 429 0 0; 27 600 600 0 3; 80 624 624 0 0; 4 810 810 0 4; 10 810 866 56 8; 63 847 847 0 4; 44 870 870 0 5; 57 915 915 0 1; 97 966 990 24 0]

[21 330 330 0 4; 37 360 364 4 5; 28 390 420 30 8; 90 421 421 0 7; 81 450 450 0 4; 74 499 499 0 0; 6 600 600 0 1; 59 731 731 0 0; 35 840 840 0 1; 30 840 855 15 2; 83 909 909 0 1; 88 939 939 0 0]

[9 270 270 0 4; 53 330 353 23 6; 26 360 360 0 8; 62 390 420 30 4; 106 431 438 7 2; 79 460 460 0 0; 8 570 570 0 2; 41 840 840 0 4; 2 840 872 32 5; 55 870 870 0 4; 61 873 875 2 2; 94 935 935 0 0]

[34 360 360 0 1; 13 360 402 42 2; 40 360 427 67 6; 87 415 415 0 5; 93 444 444 0 1; 66 585 585 0 0; 52 840 840 0 2; 105 941 941 0 0; 46 1080 1080 0 3; 99 1159 1159 0 0]

[33 360 360 0 2; 3 360 426 66 5; 86 440 440 0 3; 56 469 471 2 0; 49 840 840 0 2; 42 900 900 0 6; 102 919 935 16 4; 95 1000 1000 0 0; 1 1080 1080 0 1; 54 1219 1219 0 0]

[]

[]

Instance 56

[45 360 360 0 2; 101 445 445 0 0; 29 630 630 0 4; 34 630 648 18 8; 85 750 750 0 4; 90 797 797 0 0; 5 810 828 18 3; 15 840 862 22 6; 23 840 884 44 7; 55 840 841 1 8; 61 865 878 13 5; 11 870 890 20 7; 79 893 893 0 6; 111 905 917 12 5; 67 922 933 11 3; 71 951 951 0 0; 20 1080 1080 0 1; 76 1132 1132 0 0]

[]

[9 330 330 0 4; 8 360 360 0 6; 39 360 361 1 7; 65 418 418 0 3; 64 429 429 0 1; 95 455 455 0 0; 38 570 570 0 3; 52 600 600 0 4; 42 630 630 0 5; 94 633 633 0 2; 108 658 658 0 1; 98 688 688 0 0; 51 810 810 0 1; 10 840 840 0 4; 107 869 869 0 3; 35 870 870 0 4; 66 935 935 0 1; 91 937 967 30 0; 28 1050 1050 0 4; 84 1112 1112 0 0]

[19 360 360 0 4; 43 360 374 14 8; 99 409 409 0 4; 75 462 462 0 0; 37 810 810 0 4; 54 840 863 23 6; 41 840 864 24 7; 4 840 898 58 8; 93 878 878 0 4; 60 919 919 0 3; 110 936 950 14 1; 97 1083 1083 0 0]

[27 330 330 0 4; 22 360 382 22 7; 83 424 424 0 3; 78 462 464 2 0; 26 780 780 0 1; 25 810 846 36 4; 3 840 888 48 6; 46 870 888 18 8; 81 897 897 0 5; 82 913 928 15 4; 102 971 977 6 2; 59 1035 1035 0 0; 6 1050 1121 71 2; 33 1080 1102 22 3; 89 1155 1155 0 2; 62 1182 1191 9 0]

[21 360 360 0 2; 77 422 422 0 0; 30 630 630 0 3; 86 766 766 0 0; 40 840 840 0 2; 13 840 841 1 3; 2 840 842 2 4; 56 870 870 0 5; 14 870 872 2 7; 69 914 914 0 6; 96 931 931 0 4; 70 932 933 1 2; 58 944 944 0 1; 112 956 956 0 0; 17 1050 1050 0 4; 7 1110 1110 0 5; 73 1133 1133 0 1; 63 1300 1300 0 0]

[50 360 360 0 4; 18 420 420 0 7; 106 452 452 0 3; 74 493 493 0 0; 12 810 810 0 1; 36 840 840 0 2; 49 840 843 3 5; 47 840 846 6 8; 68 874 874 0 7; 105 892 892 0 4; 103 912 912 0 1; 92 952 952 0 0; 31 1080 1080 0 4; 87 1167 1167 0 0]

[1 360 360 0 1; 16 360 390 30 2; 48 390 390 0 3; 72 425 425 0 2; 104 444 444 0 1; 57 464 464 0 0; 53 540 540 0 4; 109 625 625 0 0; 32 810 810 0 1; 44 840 863 23 2; 100 913 913 0 1; 88 925 931 6 0; 24 1020 1020 0 4; 80 1069 1069 0 0]

Instance 90

[54 330 330 0 2; 65 360 360 0 4; 83 360 362 2 6; 144 406 406 0 4; 155 434 434 0 2; 173 448 448 0 0; 43 600 600 0 4; 61 690 690 0 8; 133 758 758 0 4; 151 786 786 0 0; 10 810 810 0 3; 90 840 840 0 6; 35 840 842 2 7; 100 865 873 8 4; 125 925 925 0 3; 180 961 961 0 0; 87 1080 1080 0 1; 79 1140 1140 0 2; 177 1181 1181 0 1; 169 1216 1216 0 0]

[30 360 360 0 3; 18 360 394 34 6; 40 390 438 48 7; 108 444 444 0 4; 120 462 476 14 1; 130 491 524 33 0; 23 690 690 0 2; 113 747 747 0 0; 70 840 840 0 4; 67 840 872 32 8; 160 890 895 5 4; 157 936 936 0 0]

[27 330 330 0 2; 33 360 360 0 5; 9 360 362 2 7; 22 360 363 3 8; 117 381 381 0 6; 112 388 388 0 5; 44 390 390 0 6; 99 435 435 0 4; 123 445 445 0 1; 134 447 447 0 0; 88 570 570 0 1; 26 600 600 0 2; 178 629 638 9 1; 116 673 683 10 0; 1 870 870 0 1; 51 900 900 0 5; 91 939 939 0 4; 141 959 964 5 0]

[85 330 330 0 1; 13 360 360 0 3; 34 360 363 3 6; 124 406 406 0 3; 86 420 420 0 5; 103 443 443 0 3; 175 497 497 0 2; 176 500 500 0 0; 20 540 540 0 1; 110 589 589 0 0; 82 600 600 0 2; 19 630 630 0 5; 109 708 708 0 2; 172 794 794 0 0; 50 810 810 0 2; 58 840 840 0 5; 42 840 841 1 6; 148 884 884 0 3; 140 914 914 0 1; 132 964 964 0 0; 32 1080 1080 0 4; 122 1199 1199 0 0]

[41 360 360 0 2; 73 360 366 6 5; 131 433 433 0 3; 163 531 531 0 0; 49 600 600 0 1; 62 630 653 23 2; 152 698 698 0 1; 139 755 755 0 0; 66 780 780 0 3; 25 810 817 7 4; 7 810 837 27 7; 156 854 854 0 4; 115 878 878 0 3; 97 930 930 0 0; 72 930 946 16 4; 162 1012 1012 0 0; 31 1110 1110 0 4; 121 1184 1184 0 0]

[28 360 360 0 3; 47 360 370 10 7; 137 428 428 0 3; 118 447 447 0 0; 71 600 600 0 4; 56 630 654 24 5; 161 694 694 0 1; 146 710 720 10 0; 39 840 840 0 2; 129 943 943 0 0; 75 1080 1080 0 1; 165 1126 1127 1 0]

[17 360 360 0 3; 29 360 375 15 4; 4 390 395 5 7; 107 414 414 0 4; 94 438 438 0 1; 80 450 450 0 4; 119 476 476 0 3; 170 544 544 0 0; 48 600 600 0 3; 37 630 630 0 6; 138 660 660 0 3; 127 678 678 0 0; 81 780 780 0 1; 14 810 810 0 5; 171 849 849 0 4; 68 900 900 0 5; 104 937 937 0 1; 158 949 959 10 0; 53 1050 1050 0 1; 45 1080 1105 25 3; 135 1148 1148 0 1; 143 1186 1186 0 0]

[52 300 300 0 4; 46 330 330 0 8; 142 406 406 0 4; 63 420 460 40 6; 136 429 435 6 2; 153 489 489 0 0; 2 600 600 0 3; 74 630 630 0 5; 92 687 687 0 2; 164 701 703 2 0; 64 810 810 0 4; 12 840 856 16 6; 3 870 887 17 8; 93 903 903 0 6; 102 914 942 28 4; 154 949 949 0 0; 38 1050 1050 0 1; 128 1093 1093 0 0]

[84 330 330 0 1; 174 430 430 0 0; 77 600 600 0 1; 167 652 654 2 0; 76 870 870 0 2; 166 975 975 0 0]

[8 360 360 0 1; 60 390 390 0 5; 55 390 418 28 6; 145 421 435 14 5; 150 453 453 0 1; 98 456 456 0 0; 16 660 660 0 2; 106 794 794 0 0; 36 840 859 19 3; 78 840 863 23 4; 126 904 904 0 1; 168 917 919 2 0; 11 930 932 2 1; 101 1001 1001 0 0]

[15 330 330 0 4; 57 360 360 0 5; 105 380 380 0 1; 147 438 438 0 0; 6 600 600 0 2; 96 665 665 0 0; 24 780 780 0 3; 5 840 840 0 7; 69 870 883 13 8; 95 907 907 0 4; 114 935 937 2 1; 159 962 962 0 0; 89 1050 1050 0 1; 179 1099 1099 0 0]

[21 360 360 0 4; 111 398 398 0 0; 59 840 840 0 3; 149 891 891 0 0]

[]

IV. Complete Results

		size	1							2							3						
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	Number	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	N vehicule	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N ve
Inial version	RL_d55	28	3356	1964	429,5	962	273		**yes**	3037	1842	233	962	501		**yes**	3454	2060	432	962	228	no	
	RL_d01	46	6709	4032	801	1875	626		no	7002	4011	803	2187	482		**yes**	7079	4337	514	2187	576	no	
	RL_d96	53	8035	4684	1019	2331	510		**yes**	7404	4302	771	2331	500		**yes**	8007	4552	1122	2331	852	no	
	RL_d30	56	6501	3900	908	1693	452		**yes**	6305	4192	637,5	1693	307		**yes**	6481	3863	1136	1482	912	**yes**	
	RL_d2	90	11125	6336	1042	3746	1029		**yes**	10618	5967	1480	3170	1112		**yes**	11293	6424	1410	3458	1111	**yes**	
Improved parameters			1							2							3						
Set3			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	N vehicu	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	N vehicule	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N ve
	RL_d55	28	3171	1901	308	962	459	5	no	3162	1931	269	962	413	5	no	3223	2004	257	5	322	no	5
	RL_d01	46	6768	3841	739	2187	344	7	no	6570	3918	776	1875	739	6	yes	6803	4014	914	1875	343	no	6
	RL_d96	53	7720	4500	596	2623	475	9	yes	7269	4080	565	2623	824	9	yes	7590	4099	868	2623	461	yes	9
	RL_d30	56	5738	3773	482	1482	1041	7	no	5960	3437	1041	1482	782	7	yes	6000	3603	704	1693	998	no	8
	RL_d2	90	10406	5898	1337	11	954	11	yes	10180	5739	982	3458	1526	12	yes	10209	5655	807	3746	1249	yes	13
Improved parameters																							
Set 1			Total Cost	Distance	Delays Cost	Nbr V Cost	Time		LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time		LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	
	RL_d01	46	6692	4002	502	2188	697		yes	6757	3996	573	2187	581		no	6718	3819	1024	175	630	no	
	RL_d30	56	6085	3744	647	1693	479		yes	6020	3657	669	1693	635		yes	5964	3517	753	1693	632		
	RL_d2	90	10577	6213	1194	3170	889		yes	10644	6148	1037	3458	725		yes	10867	6360	1337	3170			
Improved parameters																							
Set2																							
	RL_d01	46	6530	3948	706	1875	879	6	no	6719	3984	859	1875	590	6	no	6409	3824	709	1875	540	yes	6
3	RL_d30	56	6120	3726	911	1482	584	7	no	6264	3891	679	1693	475	8	no	6446	3919	833	1693	417	yes	8
34	RL_d2	90	10288	5676	1153	3458	1013	12	yes	10193	5864	1159	3170	856	11	yes	10884	6336	1377	3170	1182	no	11
			1							2							3						
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	N vehicu	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	N vehicule	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N ve
Time depende	RL_d55	28	3046	1626	458	962	414	5	no	3002	1840	200	962	415	5	no	3094	1903	229	962	369	no	5
	RL_d01	46	6380	3837	668	1875	946	6	yes	6187	3741	258	2187	746	7	no	6373	4028	469	1875	494	no	6
	RL_d96	53	7430	4195	612	2623	690	9	no	7272	4499	149	2623	644	9	no	7925	4618	514	2623	530	yes	9
	RL_d30	56	6714	4216	1016	1482	379	7	yes	6535	4137	915	1482	327	7	yes	6035	3879	674	1482	892	no	7
	RL_d2	90	10698	6154	1085	3458	1238	12	yes	11157	6355	1056	3746	467	13	yes	10970	6713	798	3458	763	yes	12
			1							2							3						
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	time lns	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	time lns	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	time
Larger operat	RL_d55	28	3201	2005	234	962	228	0,39	yes	3271	1878	431	962	411	0,36	no	2907	1766	178	962	593	no	0,2
	RL_d01	46	6949	4662	811	1875	530	28	no	7089	3921	980	2187	727	2,49	yes	6398	3640	570	2187	789	yes	30
	RL_d96	53	7118	3937	558	2623	421	161	yes	7246	4363	277	2626	627	242	yes							
	RL_d30	56																					
	RL_d2	90	10783	5949	1375	3458	1117	1080	yes	10547	5736	1064	3746	2890	2112	yes							

		size	4							5					
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	LNS	N ve	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS
Inial version	RL_d55	28	3370	2054	355	962	319	no		3289	1798	528,5	962	456	no
	RL_d01	46	7071	4040	1468	1562	530	**yes**		6865	3989	688	2187	623	**yes**
	RL_d96	53	7915	4566	726	2623	603	no		7831	4438	769	2623	652	no
	RL_d30	56	6019	3634	692	1693	358	**yes**		5545	3287	564	1693	1174	no
	RL_d2	90	10954	6195	1300	3458	1223	**yes**		10273	5830	984,5	3458	1283	**yes**
Improved parameters			4							5					
Set3			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	LNS	N ve	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS
	RL_d55	28	3367	1949	456	962	367	no	5	3118	1819	337	962	322	no
	RL_d01	46	6534	3873	785	1875	343	yes	6	6492	3819	797	1875	738	yes
	RL_d96	53	7987	4587	1067	2331	463	yes	8	7471	4330	809	2331	565	yes
	RL_d30	56	5908	3581	633	1693	680	no	8	5950	3570	686	1693	672	no
	RL_d2	90	10318	5766	1093	3458	888	yes	12	10487	5855	885	3746	794	yes
Improved parameters															
Set 1			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	LNS		Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS
	RL_d01	46	6844	4096	561	2187	628	no							
	RL_d30	56													
	RL_d2	90													
Improved parameters															
Set2															
	RL_d01	46													
3	RL_d30	56													
34	RL_d2	90													
			4							5					
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	LNS	N ve	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS
Time depende	RL_d55	28													
	RL_d01	46													
	RL_d96	53													
	RL_d30	56													
	RL_d2	90													
			4							5					
			Total Cost	Distance	Delays Cost	Nbr V Cost	Time	LNS	time	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS
Larger operat	RL_d55	28	3165	1880	332	962	502	yes	0,4	3255	2062	231	962	320	no
	RL_d01	46	6353	3710	768	1875	294	yes	169	6245	3741	628	6245	540	yes
	RL_d96	53													
	RL_d30	56													
	RL_d2	90													

		size						Totaux						
hicule			Best	AVG	dev	AVG time	in min	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	% Distance	%Delay	% Vehicule
Inial version	RL_d55	28	3037	3301,2	0,0800315	355,4	5,92333333	16506	9718	1978	4810	59%	12%	29%
	RL_d01	46	6709	6945,2	0,0340091	567,4	9,45666667	34726	20409	4274	9998	59%	12%	29%
	RL_d96	53	7404	7838,4	0,05541947	623,4	10,39	39192	18876	3937,5	8254	61%	13%	27%
	RL_d30	56	5545	6170,2	0,10132573	640,6	10,6766667	30851	22542	4407	12239	58%	11%	31%
	RL_d2	90	10273	10852,6	0,05340656	1151,6	19,1933333	54263	30752	6216,5	17290	57%	11%	32%
Improved parameters														
Set3														
	RL_d55	28	3118	3208,2	0,02811545	376,6	6,27666667	16041	9604	1627	3853	60%	10%	24%
	RL_d01	46	6492	6633,4	0,02131637	501,4	8,35666667	33167	19465	4011	9687	59%	12%	29%
	RL_d96	53	7269	7607,4	0,044483	557,6	9,29333333	38037	21596	3905	12531	57%	10%	33%
	RL_d30	56	5738	5911,2	0,02930031	834,6	13,91	29556	17964	3546	8043	61%	12%	27%
	RL_d2	90	10180	10320	0,01356589	1082,2	18,0366667	51600	28913	5104	14419	56%	10%	28%
Improved parameters														
Set 1														
	RL_d01	46	6409	6552,66667	0,02192492	669,666667	11,1611111							
	RL_d30	56	6120	6276,66667	0,02496017	492	8,2							
	RL_d2	90	10193	10455	0,02505978	1017	16,95							
Improved parameters														
Set2														
	RL_d01	46												
3	RL_d30	56												
34	RL_d2	90												
hicule														
Time depende	RL_d55	28	3002	3047,33333	0,01487639	399,333333	6,65555556	9142	5369	887	2886	59%	10%	32%
	RL_d01	46	6187	6313,33333	0,02001056	728,666667	12,1444444	18940	11606	1395	5937	61%	7%	31%
	RL_d96	53	7272	7542,33333	0,03584214	621,333333	10,3555556	22627	13312	1275	7869	59%	6%	35%
	RL_d30	56	6035	6428	0,06113877	532,666667	8,87777778	19284	12232	2605	4446	63%	14%	23%
	RL_d2	90	10698	10941,6667	0,02226961	822,666667	13,7111111	32825	19222	2939	10662	59%	9%	32%
						0								
Ins														
Larger operat	RL_d55	28												
	RL_d01	46												
	RL_d96	53												
	RL_d30	56												
	RL_d2	90												

	run	size	1							2						
			Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	N vehicule	LNS	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	N vehicule	LNS
Final version	RL_d55	28	2964	1875	127	962	293	5	no	2929	1906	61	962	388	5	no
	RL_d01	46	6430	4054	500	1875	559	6	no	6437	3968	594	1875	903	6	no
	RL_d96	53	7512	4429	460	2623	670	9	yes	7576	4140	813	2623	617	9	yes
	RL_d30	56	5715	3492	740	1482	1295	7	yes	5653	3670	501	1482	1170	7	no
	RL_d2	90	10222	5694	781	3746	990	13	yes	9575	5573	543	3458	1813	12	no

	run	size	3							4						
			Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N vehicule	Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N vehicule
Final version	RL_d55	28	3070	1804	304	962	389	yes	5	2841	1622	257	962	387	yes	5
	RL_d01	46	6436	4001	247	2187	917	no	7	6292	3483	621	2187	417	no	7
	RL_d96	53	7204	4105	475	2623	1155	no	9	7307	4202	482	2623	599	no	9
	RL_d30	56	5671	3430	547	1693	732	yes	8	5982	3582	616	1693	1278	no	8
	RL_d2	90	10523	6197	868	3458	1222	yes	12	10153	5893	513	3746	1690	yes	13

	run	size	5						
			Total Cost	Distance Cost	Delays Cost	Nbr V Cost	Time	LNS	N vehicule
Final version	RL_d55	28	2994	1695	337	962	351	no	5
	RL_d01	46	6496	3914	394	2187	813	no	7
	RL_d96	53	7295	3990	390	2914	693	yes	10
	RL_d30	56	5809	3755	360	1693	883	no	8
	RL_d2	90	10607	5984	876	3746	939	yes	13

	run	size						Totaux						
			Best	AVG	dev	AVG time		Total Cost	Distance Cost	Delays Cost	Nbr V Cost			
Final version	RL_d55	28	2841	2959,6	0,04007298	361,6	6,02666667	14798	8902	1086	4810	60%	7%	33%
	RL_d01	46	6292	6418,2	0,01966283	721,8	12,03	32091	19420	2356	10311	61%	7%	32%
	RL_d96	53	7204	7378,8	0,02368949	746,8	12,4466667	36894	20866	2620	13406	57%	7%	36%
	RL_d30	56	5653	5766	0,01959764	1071,6	17,86	28830	17929	2764	8043	62%	10%	28%
	RL_d2	90	9575	10216	0,06274471	1330,8	22,18	51080	29341	3581	18154	57%	7%	36%

10. List of Appendix

I. Summary of several algorithms for the dynamic multi-vehicle DARP 66

II. Identification for the best parameters setting for the hybrid BA-DA (BA-SA). 68

III. Best solutions with final algorithm 70

IV. Complete Results 74

11. Bibliography and References

Aldaihani, M., Dessouky, M. M. (2003). Hybrid scheduling methods for paratransit operations. *Computers & Industrial Engineering, 45*, 75–96.

Atahran, A., Lenté, C., T'kindt, V., (2014). A multicriteria dial-a-ride problem with an ecological measure and heterogeneous vehicles. J. *Multi-Criteria Decis.Anal.* (Forthcoming). http://dx.doi.org/10.1002/mcda.1518.

Baugh JW, Krishna G, Kakivaya R, Stone JR (1998) Intractability of the dial a-ride problem and a multiobjective solution using simulated annealing. *Eng Optim* 30:91 – 123.

Braekers, K., Caris, A., Janssens, G.K., (2013). Integrated planning of loaded and empty container movements. *OR Spectrum 35 (2)*, 457–478.

Braekers, K., Caris, A., Janssens, GK., (2014). Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research: Part B*. 2014; 67:166-186. doi:10.1016/j.trb.

Chassaing, M., Duhamel C., Lacomme P., Laforest C. (s.d). DARP Instances Real Life*. Limos. Part B: Methodological,* 20 (3), 243 {257.https://perso. isima. fr/~lacomme/Maxime/Real_life_instances/Real_life_instances.php}

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., Trubian, M. (1996) Heuristics from nature for hard combinatorial optimisation problems. *Int Trans Oper.Res 3*,1–21

Cordeau, J.-F., (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Oper. Res. 54 (3)*, 573–586.

Cordeau, J. -F., Laporte, G., (2003a). The Dial-a-Ride Problem (DARP): variants, modeling issues and algorithms*. 4OR: Quart. J. Oper. Res. 1 (2)*, 89–101.

Cordeau, J. -F., Laporte, G., (2003b). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transp. Res. Part B: Methodol*. 37 (6), 579–594.

Cordeau, J. -F., Laporte, G., (2007). The dial-a-ride problem: models and algorithms. *Ann. Oper. Res. 153 (1)*, 29–46.

Desrosiers, J., Dumas, Y., Soumis, F., Taillefer, S., & Villeneuve, D. (1991). An algorithm for mini-clustering in handicapped transport. *Les Cahiers du GERAD*, *G-91-02, HEC Montréal.*

Dueck, G., Scheuer, T., (1990). Threshold accepting: A general purpose optimisation algorithm appearing superior to simulated annealing. *J. Computat. Phys. 90 (1)*, 161–175.

Dumas, Y., Desrosiers, J., & Soumis, F. (1989a). Large scale multi-vehicle dial-a-ride problems. *Les Cahiers du GERAD, G-89-30, HEC Montréal.*

Freitas, A.A., (2013). Data Mining and Knowledge Discovery With Evolutionary Algorithms. *Springer Science & Business Media*, Berlin.

Gschwind, T., Drexl, M.(2019). Adaptive Large Neighbourhood Search with a Constant-Time Feasibility Test for the Dial-a-Ride Problem. *Transportation science, vol.53, No.2,*480-491. https://doi.org/10.1287/trsc.2018.0837

Guerriero, F., Bruni, M.E., Greco, F., (2013). A hybrid greedy randomized adaptive search heuristic to solve the dial-a-ride problem. *Asia-Pac. J. Oper. Res. 30 (1)*, 1250046.

Ho, SC., Haugland, D. (2004) Local search heuristics for the probabilistic dial-a-ride problem. *Tech. Rep. 286, University of Bergen.*

Ichoua, S., Gendreau, M., Potvin, J-Y. (2003). Vehicle Dispatching with time – dependent travel times. *European Journal of Operational Research. 144.* 379-396. 10.1016/S0377-2217 (02)00147-9.

Ioachim, I., Desrosiers, J., Dumas, Y., & Solomon, M. M. (1995). A request clustering algorithm for door-to-door handicapped transportation*. Transportation Science, 29*, 63–78.

Jain, S., Van Hentenryck, P., (2011). Large neighbourhood search for dial-a-ride problems. In: Lee, J. (Ed.), Principles and Practice of Constraint Programming CP 2011, Lecture Notes in Computer Science, 6876. Springer, Berlin Heidelberg, 400–413.

Jaw, J. J. (1984) *Heuristic Algorithms for Multi-Vehicle, Advance-Request Dial-A-Ride Problems. Ph.D. Thesis, Dept. of Aeronautics and Astronautics, M.I.T., Cambridge*, MA.

Jaw, J.-J., Odoni, A. R., Psaraftis, H. N., and Wilson, N. H. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Trampn. Rcs:8 Vol. 2OB. No. 3*. pp. 243–257.

Jørgensen, R. M., Larsen, J., & Bergvinsdottir, K. B. (2007). Solving the dial-a-ride problem using genetic algorithms. Journal of the Operational Research Society.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M., (2011). The irace Package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives*. 3. 10.1016/j.orp.2016.09.002.

Masmoudi, MA. Hosny, M. Braekers, K. Dammak, A.(2016). Three effective metaheuristics to solve the multi-depot multi-trip heterogeneous dial-a-ride problem. *Transportation Research: Part E*. 2016; 96:60-80. doi:10.1016/j.tre.

Melachrinoudis E., Ilhan A.B., Min H (2007) A dial-a-ride problem for client transportation in a health care organization. *Comput Oper Res 34*,742–759.

Miller, B.L., Goldberg, D.E., (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst. 9 (3),* 193–212.

Parragh, S.N., Doerner, K. F., Hartl, R.F., (2008). A survey on pickup and delivery problems. Part II: transportation between pickup and delivery locations. *J.Betriebswirtschaft 58 (2)*, 81–117.

Parragh, S.N., Doerner, K.F., Hartl, R.F., (2010). Variable neighbourhood search for the dial-a-ride problem. *Comput. Oper. Res. 37 (6)*, 1129–1138.

Parragh, S.N., Schmid, V., (2013). Hybrid column generation and large neighbourhood search for the dial-a-ride problem. *Comput. Oper. Res. 40 (1)*, 490–497.

Psaraftis, H. N. (1980). A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem*. Transportation Science, 14*, 130–154.

Psaraftis, H. N. (1983). An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows*. Transportation Science, 17*, 351–357.

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., Zaidi, M., (2005). The Bees Algorithm. Technical Note*. Manufacturing Engineering Centre*, Cardiff

Pham, DT., Castellani, M., Chen, J. (2015). A comparative study of the Bees Algorithm as a tool for function optimisation. *Cogent Engineering. 2(1):1-N.PAG*. doi:10.1080/23311916.2015.1091540

Rekiek, B., Delchambre, A., & Saleh, H. A. (2006). Handicapped person transportation: an application of the grouping genetic algorithm. *Engineering Application of Artificial Intelligence, 19*, 511–520.

Ropke, S. (2005). The pickup and delivery problem: models and optimisation algorithms [presentation from seminar guest]. Department of Computer Science, University of Copenhagen and IT University of Copenhagen. https://rasmuspagh.net/courses/CAOS/DARP.pdf

Ropke, S., Cordeau, J.-F., Laporte, G., (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks 49 (4),*258–272.

Sexton, T. (1979). The single vehicle many-to-many routing and scheduling problem. Ph.D. *dissertation, SUNY at Stony Brook*.

Sexton, T., & Bodin, L. D. (1985a). Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Science, 19*, 378–410.

Sexton, T., & Bodin, L. D. (1985b). Optimizing single vehicle many-to-many operations with desired delivery *times: II. Routing. Transportation Science, 19*, 411–435.

Uchimura K, Saitoh T, Takahashi H (1999) The dial-a-ride problem in a public transit system. *Electron Commun Jpn 82,* 30–38.

Urra,E. Cubillos, C. Cabrera-Paniagua,D.(2014).A hyperheuristic for the dial-a-ride problem with time windows. *Mathematical problems in engineering, volume 2015*, Article ID 707056, 12 pages. https://doi.org/10.1155/2015/707056

Zidi,I. ,Mesghouni,K., Zidi,K. Ghedira, K. , 2012 . A multi-objective simulated annealing for the multi-criteria dial a ride problem. *Engineering Applications of Artificial Intelligence, Volume 25, Issue 6*, 1121–1131, https://doi.org/10.1016/j.engappai.2012.03.012.

Executive summary

The dial a ride problem (DARP) is a specific transportation problem. The objective is to optimise the planning of a collection of trips made by a fleet of vehicles. Those trips aim to satisfy requests from users while meeting various constraints. The cleared example is the transportation of the elderly or disabled people. The problem has received increase interest in recent years as the demand for services that can be solved through DARP is rising. Due to the aging of the population, this demand will be even bigger in the future. Good optimisation tools are needed to respond to it.

In this thesis, I review the different heuristics to solve DARP before implementing a hybrid Bee Algorithm with Deterministic Annealing. The different creation steps are described before the algorithm is tested on several instances. The results are then used to explore possible improvements to the algorithm. Several real-life problems are discussed along the way.

Keyword: Vehicle routing problem, Dial a ride problem, Optimisation, Hybrid metaheuristic, Bee algorithm, Deterministic Annealing

Word count = 18942