Master Operation Research, Combinatorics and Optimization
ORCO
Master of Science in Computer sciences at Grenoble
Master Computer sciences / Master Applied mathematics

Decision Support System for carpooling to reduce car traffic in Grenoble area

Romain NAVARRO

June, 28th 2018

Research project performed at G-SCOP, under the supervision of:
Prof. Van-Dat CUNG, G-SCOP & UGA
Defended before a jury composed of:
Prof. Nadia BRAUNER
Prof. Van-Dat CUNG
Dr. Vassilissa LEHOUX
Prof. Matej STEHLIK

June 2018

Abstract

The problem of carpooling is a problem of satisfaction for users who would like to have as little inconvenience as possible. We can cite some of the inconveniences of current carpooling systems, such as inadequate schedules, long detours, or not enough flexibility, home-to-work trips.

We have studied the possibilities of minimizing these inconveniences, and here are the main points:
Our system offers a daily solution to avoid long-term commitment problems and to have more flexibility.
Our system allows users to have different places of work, either if the user does not have a fixed place of work, or if he wants to go somewhere before going home. The user can also have several places of residence within a single day, if he does not want to return to the same place. Thus, the user can have one origin and destination in the morning, then another origin and destination in the evening.

Our objective is to create user groups to minimize the distance travelled by users while maximizing vehicle occupancy.
Furthermore, we will study the impact of these inconveniences on the possible solutions, by varying the possible advance at work, as well as by varying the authorized detour lengths.

Key-words: optimization - multi-destination daily carpooling problem - home to work - return management

Table of contents

— 1 — Introduction

Carpooling is subject to intense research, yet many users continue to take their personal car. Private car use accounts for the largest part of kilometers traveled and is considered one of the most important contributors to air pollution [LHCY08]. Promoting eco-friendly transportation modes (such as public transportation, bicycle, walking, or carpooling) is becoming more and more frequent. However, even with increasing environmental awareness and concern, many road users are still car-dependent, either by choice or constrained by circumstances [Str07]. These car-dependent users therefore have no access to any other means of transportation. The carpooling is the most suitable alternative [DG16].

To position ourselves we will talk about different types of carpooling.

1.1 Object: The kinds of carpooling

We will start by defining what carpooling is. Definitions vary considerably in the literature, nevertheless we can form a general definition of the idea behind these terms. Carpooling is the use of private cars, unlike car-sharing which uses cars assigned to a system. We can define it by forming groups of people, say pools, each sharing the same car. These pools are on average formed by more than one person per car. The people of the pool participate, by various means, in the expenses generated by the trips.
There are many systems for connecting people, such as websites that ask users who go from one city to another to enter their route, or universities and companies themselves forming groups of people based on their own data.

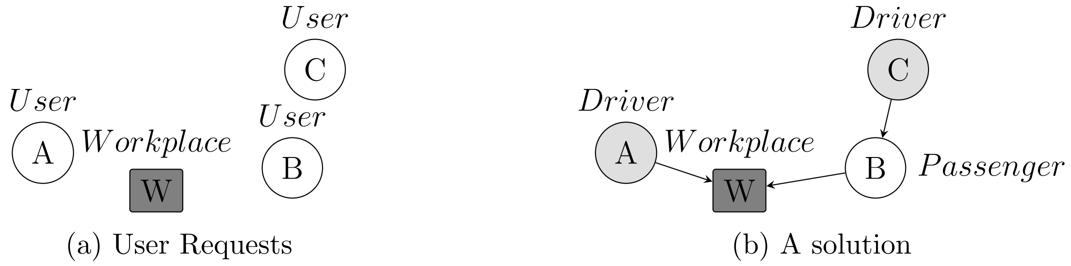
Carpooling is studied since the seventies [Woh70] [Sag74], and two types of carpooling problems have been defined since the 2000s [Kot04] [MCH04] [CdLHM04]. There are significant differences depending on the literature chosen regarding the definition of these two groups.

The Daily CarPooling Problem **DCPP** concerns the formation of pools, possibly different each day. We can say that the pools of each day are independent of the elders.

The Long Term CarPooling Problem **LCPP** concerns the formation of regular and stable pools over a given period. Some articles talk about the variation of who takes his car in each pool formed [Guo12].

Difference between DCPP and LCPP

Each Figure below represents (a) the requests and (b) a proposed solution. Each circle represents a request from a user wishing to go to work. The work is represented by a dark grey square.
In the solution, users who take their car, thus becoming drivers, are represented by a light grey circle. The arches represent the path taken by the driver.
The following Figures show examples to highlight the difference between the DCPP and the LCPP over **two consecutive days**.

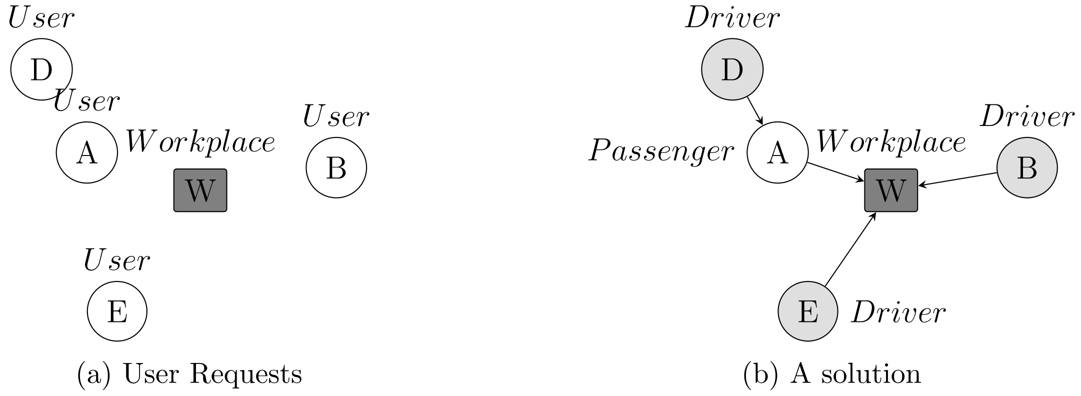


(a) User Requests

(b) A solution

Figure 1.1: DCPP: First day

Figure 1.1 shows a possible instance of the DCPP, with a set of requests and a solution.



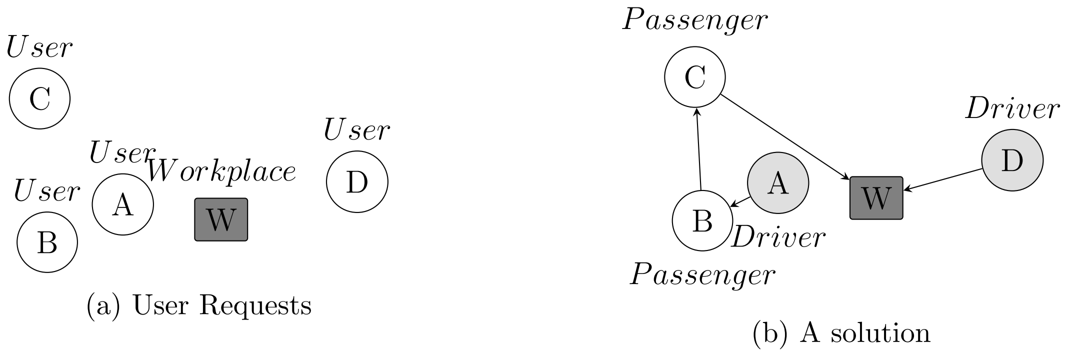
(a) User Requests

(b) A solution

Figure 1.2: DCPP: Second day

Figure 1.2 represents the following day of the Figure 1.1. We can see that there are no longer the same users. Some have not renewed their requests, and some new requests must be taken into account. Thus, we can see that the user *A* becomes a passenger on the second day, whereas he left to work alone on the first day. The user *B* was a passenger on the first day, and goes to work alone on the second day.

Now let's see the LCPP examples.

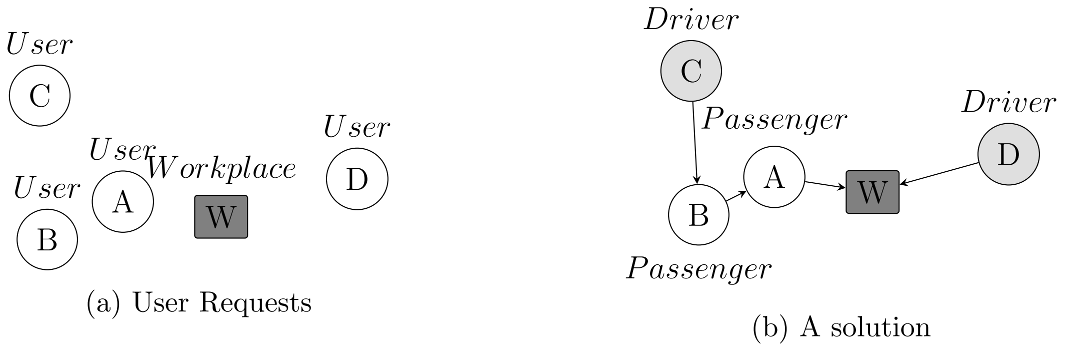


(a) User Requests

(b) A solution

Figure 1.3: LCPP: First day

Figure 1.3 concerns the LCPP, with the same pattern as in Figure 1.2.



(a) User Requests

(b) A solution

Figure 1.4: LCPP: Second day

In Figure 1.4, the regularity is put forward: we have the same users as in Figure 1.3. The pools have remained, the solution is to vary the drivers. Changes may obviously occur, however the goal remains to maintain a certain consistency.

Now that we know what carpooling is, we will see what are the issues of carpooling.

Issues of the carpooling

Carpooling has advantages for the city and for users, we can cite the reduction of traffic jams, pollution and accidents. It also leads to freer parking, savings and more user-friendliness [BGG+17].

By analyzing a few articles, we have compiled lists, representing the advantages and disadvantages for carpool users, as well as the advantages for companies that their employees make the trip together [HYB00] [OB04] [YC11] [BCCL11].

Advantages for the user:

- Splitting of expenses
- Compared to public transports
  - No waiting
  - Less time
  - No walk

- Less need to get a special car to work
- Reduce traffic congestion / traffic volumes
- Reduce noise levels
- Reduce air pollution (vehicles emission)
- Fewer accidents
- Fuel savings

Disadvantages for the user:

- Flexible hours: not convenient
  - Spread of flextime
- Increase travel time
  - Pick up
  - Deliver
  - Mutual waiting
- Loss of independence
- Loss of privacy
- Loss of self-reliance
- Anonymity of the vehicle creates a more comfortable social climate
  - Public cars

Advantages for the companies:

- Maximize the use of employee parking
- Sociability between the employees
- Reduce stress in driving to work
- Improve the company image

From this list, we can mention the disadvantages on which we will work, namely flexible working hours, travel time and user independence.
Still according to the articles, [HYB00] [OB04] [YC11] [BCCL11] we can separate the **issues** into different categories:

Environmental issues related to reducing the consumption of non-renewable energy (oil) and reducing greenhouse gas emissions.

Structural issues related to the reduction in the number of vehicles on the roads and developments related to parking. Carpooling facilities are becoming more common, such as free outdoor parking for carpoolers, near transportation or meeting points for carpoolers. If some structures are no longer useful, such as a three-lane road, the city would build new structures for the sustainable development of the city, such as green spaces and cycle paths.

Economic issues with the reduction of the cost of road maintenance for the city, and the cost of vehicle wear and tear for the user.

With the collection of data on carpooling, we would observe traffic flows in order to implement new solutions, such as the development of a new subsidiary road in a place with many companies. There are other viable solutions, such as traffic light management or the creation of new roundabouts.

We have the background to define the context in which the subject of our report is set.

Context of the report

In this report, we are interested in carpooling in France, more particularly in the case of the Grenoble conurbation and its surroundings. The French use carpooling less than their European neighbours, the reasons they give are that they have better **control of their schedule**, they arrive **on time at work**, and it is **faster** [DG16]. In response we have set up some ways to take into account these reasons.

With regard to the **control of the schedule**, we decided to make a system based on **daily carpooling**, to be able to change the schedule of the next day.
We also manage the **return trip from work**, to be closer to the user's schedule.
We also allow the user to have **different origins and destinations in the morning and evening**.
Thanks to these specificities, users benefit from greater flexibility and a more respected schedule.

Regarding the arrival **on time at work**, we set up a **time window** to **not be too ahead at work**: it is a time window between zero and a fixed time. We will explain it more in detail later.
The return trip has a similar time window system, in order to **avoid the user to wait too long** after finishing his job.

Finally, we set up a management of the travel time, to define either a **percentage of the trip allowed time**, or a **fixed time**, or **both**.
For example, on a user's 100-kilometre home-work trip, if we allow a 10 per cent detour and add 10 kilometres of detour, the result will be 100 kilometres plus 10 kilometres plus 10 kilometres, which is 120 kilometres of travel allowed for that user.

To reduce traffic congestion in Grenoble, the study focuses on the home-work commute.

We will give more details, with illustrations of examples in the presentation of the problem 3.1. To understand the current situation, we will talk about smartphone applications that are currently on the French market.

Existing applications

It is important to talk about existing applications for home-to-work trip because we need to understand the specificities of existing systems.
The carpooling market from one city to another is owned by BlaBlaCar [Dio].However, when it comes to carpooling from home to work, their subsidiary BlaBlaLines has competition. We will quickly see what characterizes each of the flagship **applications** of the moment.

BlaBlaLines links users according to their schedule, allowing a detour of at most fifteen minutes for a trip. Ultimately, they would like to set up a system of ”lines” on busy roads, where drivers receive matching requests to their schedule and their road, and have the choice to accept or ignore it [Web17]. Their goal is to have as much flexibility as possible with no long-term commitment .

Klaxit mainly relies on partnerships with communities and companies. Its principle is to find regular close carpoolers [ARN18]. It fits with people who often have the same hours within the same company.

Karos combines car and public transport. In addition, the application will learn from your usual routes to offer you matching carpoolers [Web16]. it fits with people who periodically do the same circuit.

Each of these applications have theirs advantages and theirs disadvantages, with customer types specific to each. However, the market seems to be able to offer more in view of the very low occupancy rate per car during home-to-work journeys [Web08].
In addition, the French government is implementing the reimbursement of home-to-work journeys during strike periods, which is beneficial for both users and applications [Web18].

Having the necessary background for understanding the report, we will see the order of chapters in the next section.

Conduct of the project

Here is an introduction to the chapters and the guideline followed throughout the report.

First, the positioning in the literature of our subject will be highlighted. The existing will be presented, as well as the contribution to the research of our problematic.
Secondly, the problem will be explained in detail, with illustrations of the original concepts that we bring in relation to the existing.
Third, the data model and the mathematical model used in this internship will be presented. We will explain the constraints and data used, in order to understand the resolution method used.
Finally, we will see how the data was generated and/or recovered. We will initiate the scenarios, the expected results, then we will present and analyze the results obtained.

— 2 —
State of the art

Carpooling is a vast and varied field, and the existing articles concern as much specific points as general cases.

First, we will define the class of the problem.

2.1 Class of the problem

In order to better understand the state of the art, we will define the problem classes that best correspond to our problem:

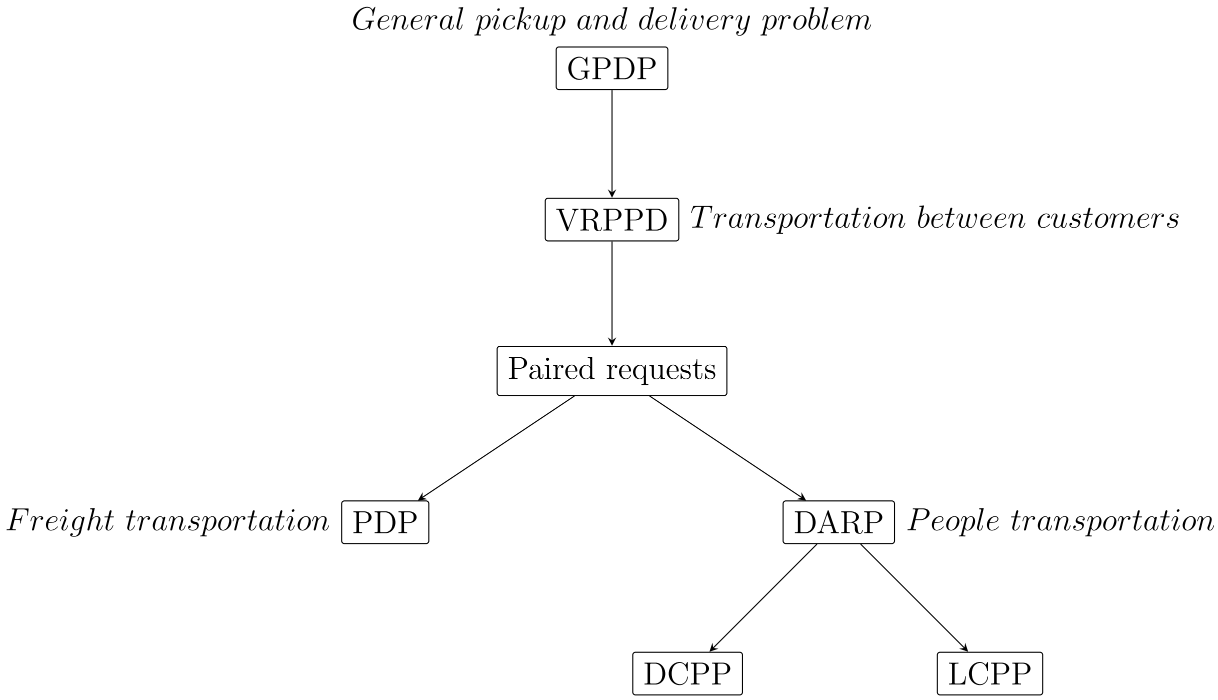


Figure 2.1: Problem classes tree

Figure 2.1 is the problem classes tree, each of these problems defined below. The definitions are inspired by [PDH08] and [Guo12].

- The General Pickup and Delivery Problem **GPDP**:
  There is a set of routes built to satisfy the transport requests.

A fleet of vehicles is available to get to the roads, where each vehicle has a capacity, a starting point and an ending point.

Each request specifies the size of the load to be transported, the origin and destination of that load.

- Vehicle Routing Problem with Pickup and Delivery **VRPDP**:

  Contraction between Vehicle Routing Problem **VRP** and Pickup and Delay Problem **PDP**.

  It concerns the transportation to customers, and between customers.

- Vehicle Routing Problem **VRP**:

  Either the origin or the destination of the vehicles are depots.

- Pickup and Delay Problem **PDP**:

  Requests specify a single origin and a single destination, and vehicles leave a repository and return to a repository.

- Dial-A-Ride Problem **DARP**:

  Pickup and delivery used for people transportation.

  Take into account the user inconveniences.

  Users formulate transport requests from an origin (pick-up point) to a destination (drop-off point) [YC11].

- **PDP** and **DARP** are **Paired**:

  Each request has an origin-destination pair.

DCPP and **LCPP** [ref: Introduction] are variants of the **DARP**, where each person taking his car is a deposit of his vehicle.

All these problems can be treated with the management or not of time windows. In the case of **DARP**, and therefore **DCPP** and **LCPP**, this management is included.
We manage people daily with our system, so **we are in the case of a Daily CarPooling Problem DCPP**.

Now that we have defined problem classes, let's speak about resolution methods: There are either heuristics methods, or exact methods, or a combination of both(hybrid).

2.2 Resolution methods

In this section will be presented some existing articles about the subject.

2.2.1 Exact methods

Exact methods are objectives and constraints giving an exact solution on a linear program.
Mr. Cordeau [Cor06] [YC11] is solving a DARP with a Branch-and-Cut linear program. *MDCPP* is an extension designating the consideration of multi-destination, there are several places of work instead of one, and Mr. Guo [Guo12] presents it in his chapter 5. In a survey[PDH08] the differences in constraints between a PDP and a DARP will be shown.

2.2.2 Heuristics

The management of many data is not possible for problems that are not solvable in a polynomial time. This is why heuristic algorithms have been designed to obtain viable solutions on large instances.

Genetic
The principle of Genetic Algorithms **GA** is based on three phases:
Natural selection: Selection of individuals most likely to obtain the best results.
Crossing: Crossing of individuals with a probability between 0 and 1.
Mutation: Probability of mutation of an individual between 0.001 and 0.01.
Mr. Huang uses this algorithm in cloud computing which consists in harnessing the computing or storage power of remote computer servers via a network [HJL15].
There are variants to these algorithms, such as GGA and AGA:
Guided Genetic Algorithm **GGA** is a Genetic Algorithm Oriented for Selection and Crossing [Guo12]. In Multi-Agent Self-Adaptive Genetic Algorithm **AGA**, each agent shares its progress with other agents. Learning mechanisms guide the genetic structure in order to have more appropriate choices. [Guo12]

Tabu
The principle of Tabu Algorithms **TA** is to find local minimums, save their position in a queue so as not to return to an already explored position. It is used in particular by Mr. Gendreau [GLS96], by Mrs. Li and Lim [LL03], and by Mr. Cordeau [CL03].

Simulation Based Approach
Use the K-averaging algorithm to form groups [CV09].

Neighborhood Search Variable
Search among ”far-off” for the best local solution [PHDR04].

Clustering Ants Colony
Forms clusters during the construction of the ”paths” of agents (ants) [ML04] [Guo12].

2.2.3 Hybrid methods

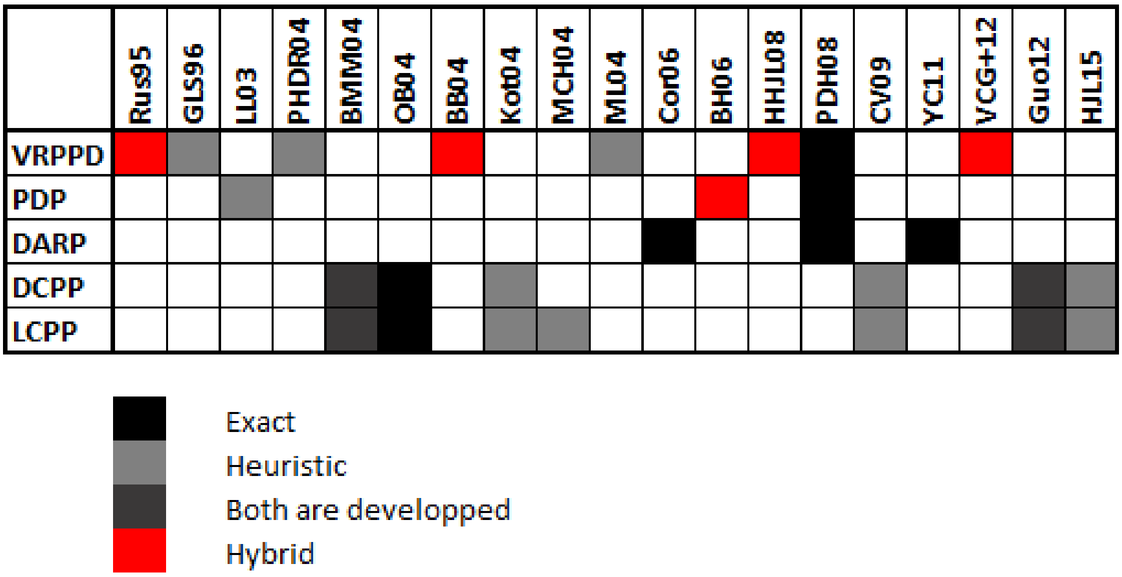
Hybrid methods make it possible to associate a heuristic method with an exact method, two heuristics or two exact methods, in order to have a good compromise between different objectives.

In the article Hybrid Heuristics for the Vehicle Routing Problem with Time Windows[Rus95] both tour construction and local search tour improvement heuristics are developed. In the article A parallel hybrid genetic algorithm for the vehicle routing problem with time windows[BB04] a parallel version of a new hybrid genetic algorithm for the vehicle routing problem with time windows is presented. The route-directed hybrid genetic approach is based upon the simultaneous evolution of two populations of solutions focusing on separate objectives subject to temporal constraint relaxation. While the 1rst population evolves individuals to minimize total traveled distance the second aims at minimizing temporal constraint violation to generate a feasible solution. In the article A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows[BH06], the first stage uses a simple simulated annealing algorithm to decrease the number of routes, while the second stage uses Large neighborhood search (LNS) to decrease total travel cost. In the article A hybrid genetic algorithm for the multi-depot vehicle routing problem[HHJL08], to deal with the problem efficiently, two hybrid genetic algorithms (HGAs) are developed in this paper. The major difference between the HGAs is that the initial solutions are generated randomly in HGA1. The Clarke and Wright saving method and the nearest neighbor heuristic are incorporated into HGA2 for the initialization procedure. In A hybrid genetic algorithm for multidepot and periodic vehicle routing problems[VCG$^+$12] Another metaheuristic combines the exploration breadth of population-based evolutionary search, the aggressive-improvement capabilities of neighborhood-based metaheuristics, and advanced population-diversity management schemes.

Mr. Baldacci propose both an exact and a heuristic method for the carpooling problem, based on two integer programming formulations of the problem. The exact method is based on a bounding procedure that combines three lower bounds derived from different relaxations of the problem. A valid upper bound is obtained by the heuristic method, which transforms the solution of a Lagrangean lower bound into a feasible solution [BMM04].

2.3 Summary of the state of the art

To summarize and relate to our problem, we will design a decision support system that will determine the pools used each day. We are in the case of a daily carpooling problem **DCPP**. We will also develop an **exact resolution method**. Then we will have **heuristics** to manage more users. We will therefore form **hybrid** techniques to have solutions on larger instances.



	Rus95	GLS96	LL03	PHDR04	BMM04	OB04	BB04	Kot04	MCH04	ML04	Cor06	BH06	HHJL08	PDH08	CV09	YC11	VCG+12	Guo12	HJL15
VRPPD	🟥	▨		▨			🟥			▨			🟥	■			🟥		
PDP			▨									🟥		■					
DARP											■			■		■			
DCPP					▩	■		▨							▨			▩	▨
LCPP					▩	■		▨	▨						▨			▩	▨

■	Exact
▨	Heuristic
▩	Both are developped
🟥	Hybrid

Figure 2.2: Existing references on the various problems

This table 2.2 shows whether articles offer solutions for general problems or more specific problems. Existing research provides generic models to cover as many problems as possible.

Other articles are based on this research to go further on a specificity. Carpooling problems have completely different solutions depending on the user satisfaction criteria taken into account.

There are already articles and systems dealing with DCPP, but people in France do not use carpooling much compared to other European countries, and one of the main reasons is is that they want to keep flexibility.
The originality brought in this report is the **management of return trip** in the specific case of **home-to-work trips**.
In addition, we also take into account a possible **return trip different** from the home-to-work trip.
Then we will study the peculiarities of the city of **Grenoble** to establish heuristics, like using motorway exists as deposit points(**Satellites**).

We will now present the problem in more detail.

— 3 —

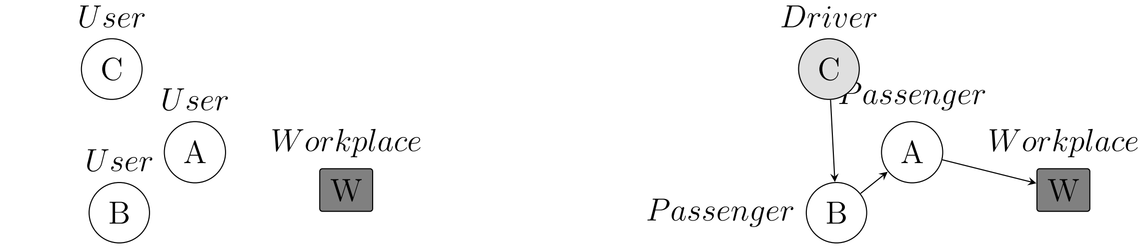
Presentation of the problem

Users wishing to carpool must, for each request, enter their vehicle capacity, address, work address and work schedule. These are the classic features of carpooling problems.

Our DCPP is located in the VRP branch, and one of the most common objectives of the VRP is to minimize the total travel cost.

3.1 Classical characteristics

The problem is to get users to work.

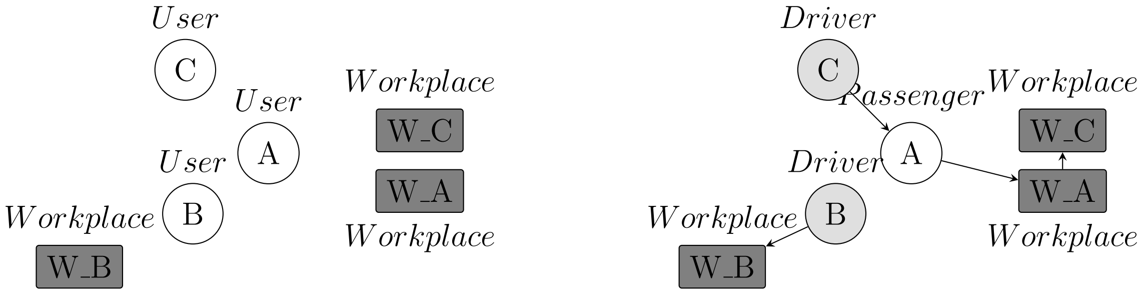


(a) User requests

(b) A solution

Figure 3.1: Same workplace

In Figure 3.1, we have an example of passengers who want to get to work. In addition, we want every user to go to their own workplace.

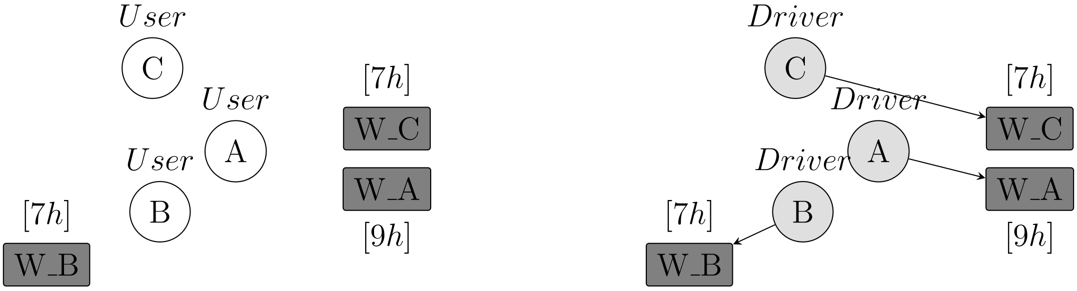


(a) User requests

(b) A solution

Figure 3.2: Different workplaces

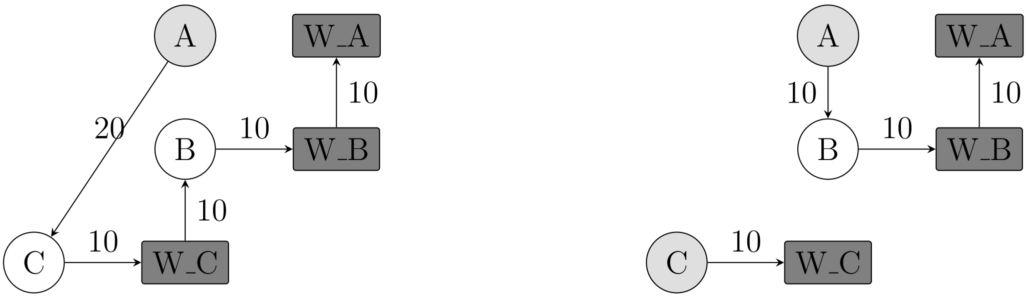
In Figure 3.2, the user's work location B is in the opposite direction to the user's work locations A and C, the separation into two pools is evident. However, another parameter is added to the problem, the management of time windows.



(a) User requests (b) A solution

Figure 3.3: Different time windows

In Figure 3.3, each user has one hour of work at which he starts working. The user A's work starts later than the user C's work, which forces each user to take his car.



(a) Total cost = 60 (b) Total cost = 40

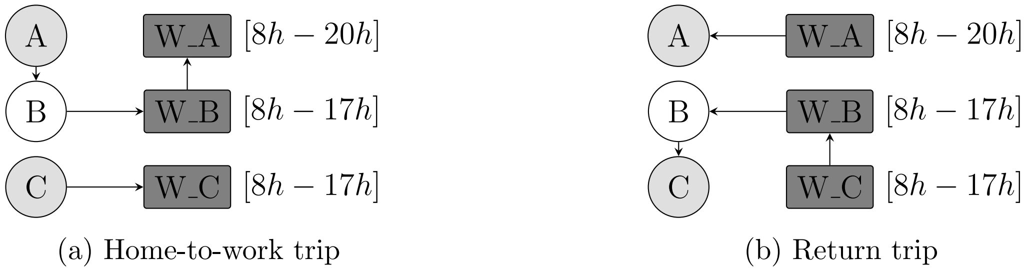
Figure 3.4: Minimization of the costs

On each arc is represented a distance / cost of fuel to go from one peak to another. In the Figure 3.4, the solution proposed in (b) is more optimal than that proposed in (a) for the same data set, since the total distance cost is lower.

3.2 Original characteristics

Return management

Return management remains an original criterion in the DCPP because Mr. GUO uses it in the context of the LCPP [Guo12], and the objective of the LCPP is to have stable pools, with the same people for the home-work trip and for the corresponding return trip. Although we do not have this constraint in the DCPP, we can have different pools.



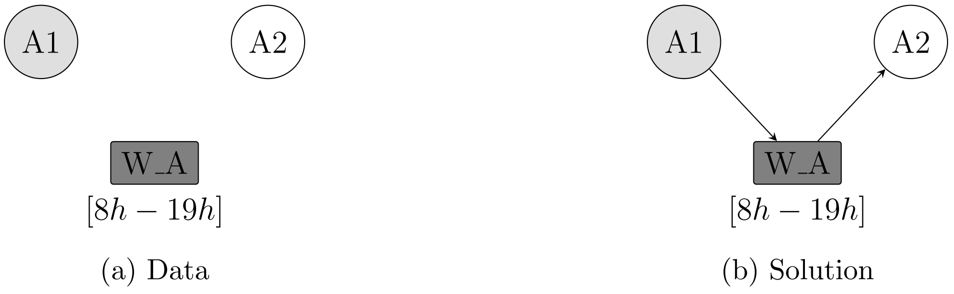
(a) Home-to-work trip (b) Return trip

Figure 3.5: Return management

Figure 3.5 represents a home-work trip (a) and the corresponding return trip (b). We can see that the composition of the vehicles changes, the schedules allowing it.

An user with multiple homes and workplaces

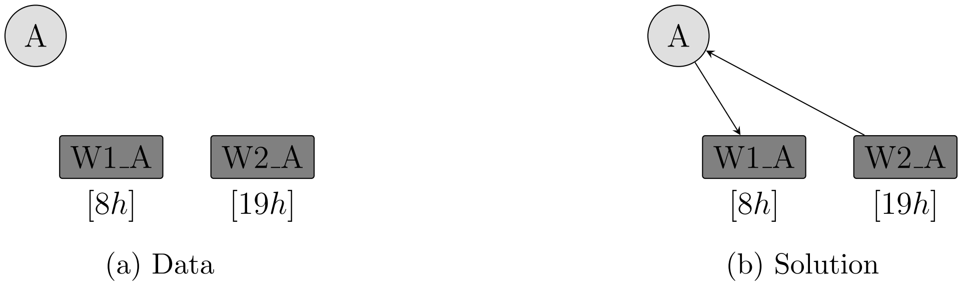
One of the other originalities of the basic problem is to have places of return from work different from those where we go to work. This is useful for example for someone who has a job where he has to change places during the day, or if someone wants to go to an activity after work. The purpose of this feature is to give more freedom to people who want to carpool.



(a) Data (b) Solution

Figure 3.6: Multiple homes

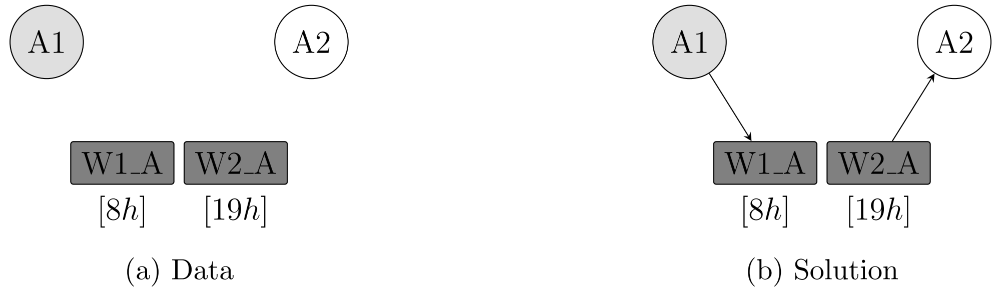
In Figure 3.6 we can see a way to materialize the different homes of the same person with $A1$ corresponding to the home address of departure, and $A2$ corresponding to the place of return (a). The proposed solution path for this person going after work to his second home address is shown in (b).



(a) Data (b) Solution

Figure 3.7: Multiple workplaces

Figure 3.7 shows the notation used when a person has multiple workplaces (a) using the same numerical nomenclature, and the corresponding solution (b). Note that there is no arc between the two places of work because the path from one to the other is independent of our system.



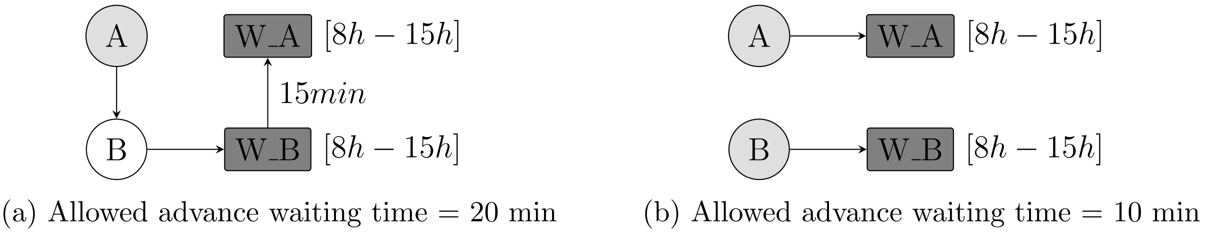
(a) Data (b) Solution

Figure 3.8: Multiple homes and workplaces

Figure 3.8 represents the union of the two previous examples. The user makes the $A1 - W1_A$ trip in the morning, then makes the $W2_A - A2$ trip in the evening.

Another kind of time windows

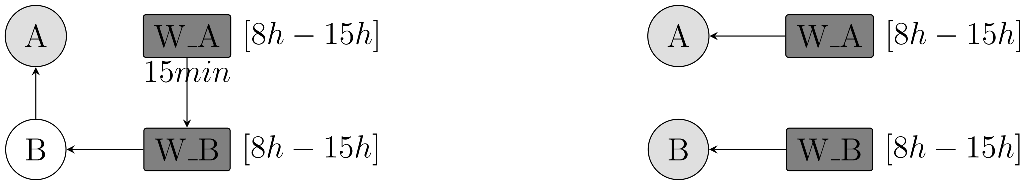
Another originality is the way we manage time windows. Indeed, it corresponds to an authorized advance time before the beginning of the work. The same thing on the way home, with an authorized waiting time at work after the end of work.
We have time windows where the user must leave home in this interval:
[work start time + maximum travel time, work start time + direct travel time]
and time of arrival at work in that interval:
work start time - time allocated in advance, work start time].



(a) Allowed advance waiting time = 20 min (b) Allowed advance waiting time = 10 min

Figure 3.9: Arrival to work waiting time

In Figure 3.9 we can see that it takes 15 minutes to travel between the two workplaces. If it is allowed to have an advance waiting time of 20 minutes (a), then a user can drop the other one 15 minutes in advance, and has time to get to work on time. If, on the other hand, there is only 10 minutes of waiting time allowed, the solution is that everyone takes his car, so that there is no user too early.



(a) Allowed waiting time after finishing work= 20 min

(b) Allowed waiting time after finishing work = 10 min

Figure 3.10: Departure from work waiting time

Figure 3.10 shows this time the return from work. This follows the same pattern as Figure 3.9, by allowing more or less waiting time after work, the solution can also evolve.

Use of satellites places

Satellite locations are places where people can be dropped off and picked up, close to a place such as home or workplace. This can be useful to the problem, in order to have more similar data and perhaps to leave more possibilities of results. For example a motorway exit close to many workplaces within 1km walk can be a good satellite point.

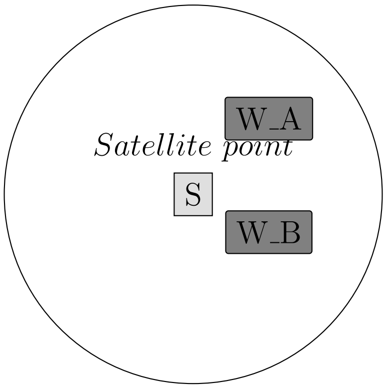


Figure 3.11: Satellite range

In Figure 3.11 we can observe a satellite point that can be used as the same point instead of using both workplaces.

We will now see in which complexity class we can put this problem.

3.3 Complexity of the problem

To prove the complexity of the problem, we will take another problem whose complexity is known, and then prove that our problem is at least as difficult.

3.3.1 Hamiltonian Path Problem

A Hamiltonian path is a path that visits each vertex of the graph exactly once.
Let's consider a generic Hamiltonian Path Problem **HPP**:

Let a graph $G = (V, E)$, with weighted edges $w : E \mapsto \mathbb{R}$, and a bound B.

Is there a Hamiltonian path in G whose sum of weights is less or equal than B ?
This problem is known to be of **NP-hard** complexity.

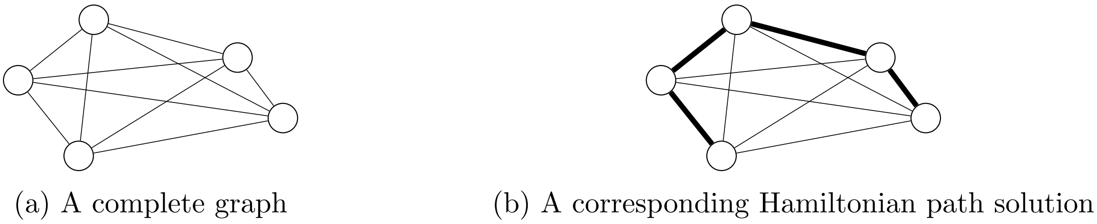

(a) A complete graph (b) A corresponding Hamiltonian path solution

Figure 3.12: Example of a Hamiltonian path in a graph

Figure 3.12 represents an example of the Hamiltonian Path Problem.

Now that we have defined the NP-hard problem, let's define our carpooling problem.

3.3.2 CarPooling Problem

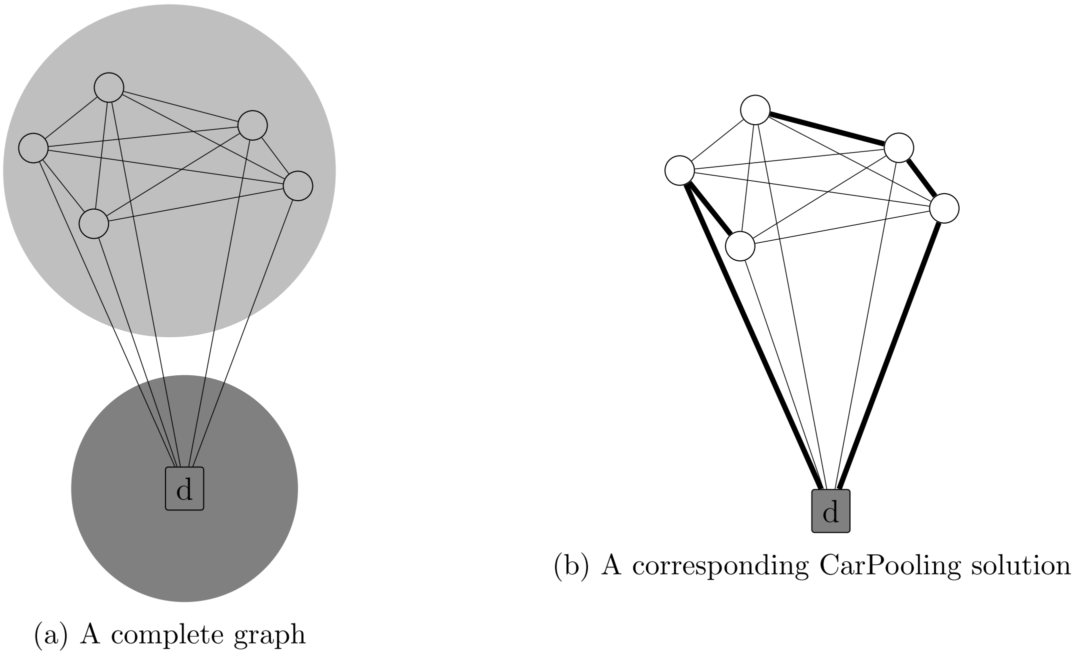
Let's consider the following generic CarPooling Problem **CPP**:

Users of a carpooling system issue requests of the form (O_i, D_i) corresponding to the origin and destination of the user i. Each user can be either passenger or driver.
The origins and destinations correspond to the vertices of a graph.

Consider a graph $G' = (V', E')$, where V' is the union between the vertices origins and the vertices destination $V' = (O \cup D)$, and E' corresponds to the edges between these vertices.

Each of the origin vertices has a weight corresponding to a vehicle capacity $c : O \mapsto \mathbb{N}$.

Each edge has a weight corresponding to a distance between the two vertices it relies $w' : E \mapsto \mathbb{R}$.
The goal is to minimize the sum of distances so that each user goes from his origin to his destination.



(b) A corresponding CarPooling solution

(a) A complete graph

Figure 3.13: Example of the CarPooling problem

In Figure 3.13 (a) we can see two areas:

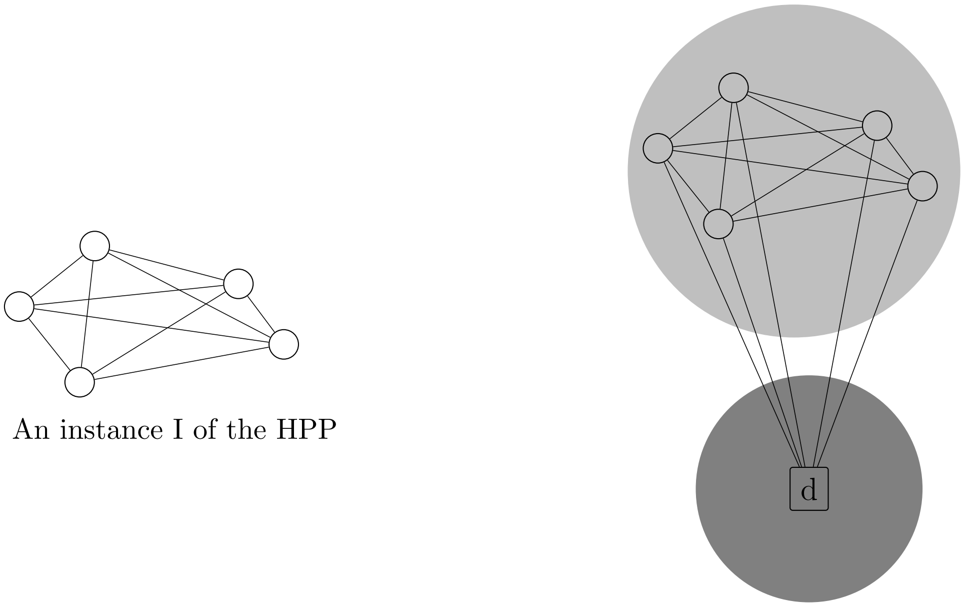
The lightgrayed zone corresponds to all the origins of the CPP.

The grayed zone corresponds to all the destinations of the CPP, for more simplicity, we gather all the destinations at one point d.

Let's understand the correspondences between the two problems.

3.3.3 Correspondences between the problems

In both cases, we are in complete charts. The graph follows the triangular inequality, the weight being related to the distance in the graph.
For any instance **I** of the HPP, there is an instance correspondence **I'** of the CPP.



(a) An instance I of the HPP

(b) An corresponding instance I' of the CPP

Figure 3.14: An example of correspondences between a HPP and a CPP

As we can see in the Figure 3.14 each vertex of instance **I** corresponds to a vertex in the set origin of instance **I'**.
For each possible instance of the HPP we have for the CPP:

$O = G$
$D = \{d\}$
$c_v = n = |V| \qquad \forall v \in O$
$w_{ij} = w'_{ij} \qquad \forall i, j \in V$
$w_{vd} = W \qquad \forall v \in O, \forall d \in D$
$W \geq B$
$B' = W + B$

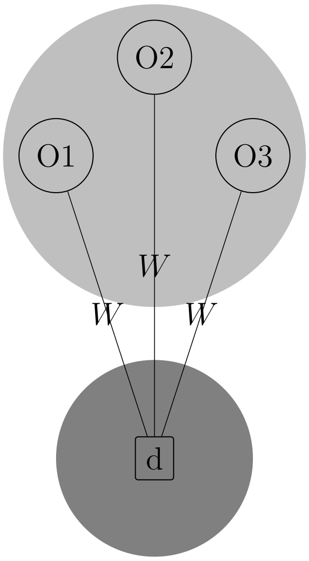


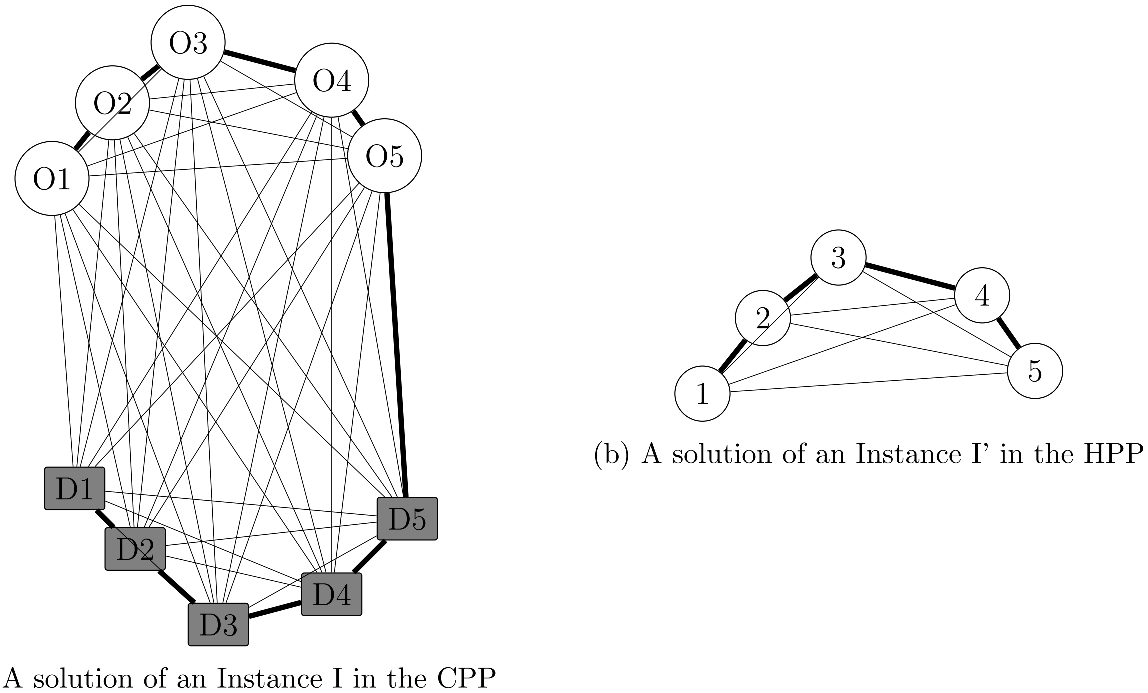
Figure 3.15: Distance between origins and destinations W

In the Figure 3.15 is represented the distance W, which is the same regardless of origin.

In order to prove that the CPP is NP-hard, we must prove the following equivalence:
I is a yes-instance for the HPP *is equivalent to* **I'** is a yes-instance for the CPP.

3.3.4 Proof of one way

Let's prove it in the way **I** is a yes-instance for the HPP if **I'** is a yes-instance for the CPP.



(a) A solution of an Instance I in the CPP

(b) A solution of an Instance I' in the HPP

Figure 3.16: Example of a solution to CPP in the HPP

If we only have one car and remove the Origin-Destination W trip, then our CPP solution are HPP solutions as shown in Figure 3.16.

Our instances of the CPP can only have **one vehicle** as a solution:
Let's see the case where there are two vehicles going to the destinations.
If we take two vehicles then we have in our solution at least twice the distance W.
We know $B < W$, then we have $B' \geq 2W > W + B$, and that's impossible because $B' = W + B$.
This counterexample proves that we can only have one vehicle traveling all the vertices.

Moreover, we have to prove this is the same solution to minimize weight in both problems.
As we know that each origin is exactly at the distance W of destinations, no matter the place of origin chosen to go to destination, we will add W to it.
So minimizing B' is equivalent to minimize the distance between the origins B.
We proved **I** is a yes-instance for the HPP if **I'** is a yes-instance for the CPP.

3.3.5 Proof in the other way

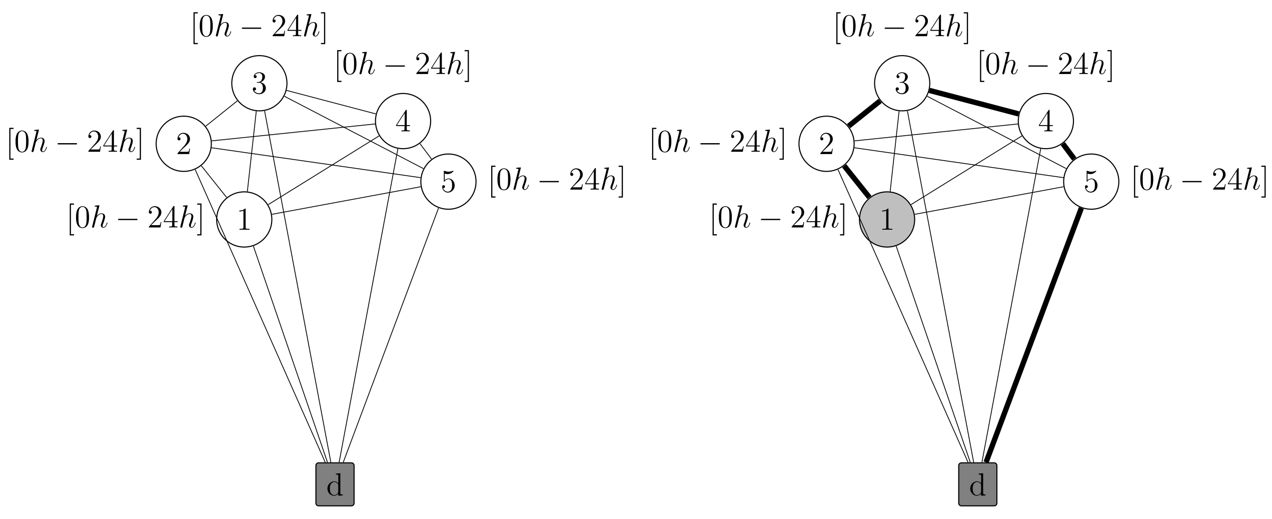
Now that we have proved a way, let's prove that **I'** is a yes-instance for the CPP if and only if **I** is a yes-instance for the HPP.
Let's take a solution of the HPP. If we add an edge from one of the extremities of the path to the destination, then we have a solution for the CPP.
The choice of the added edge does not change anything to the solution since each origin is at equal distance W from the destinations. It must be at one extremity to comply with the CPP.
As far as the number of vehicles is concerned, there can only be one since $B' = W + B$, as shown in section 3.3.4.

Now let's see if other constraints added to the problem influence the complexity.

3.3.6 Other characteristics

Time windows

The time windows associated with each vertex is not a problem because we can recreate any HPP instances where these constraints do not change the complexity of the problem.



(a) Vertices of the instance (b) A solution

Figure 3.17: CPP with time windows

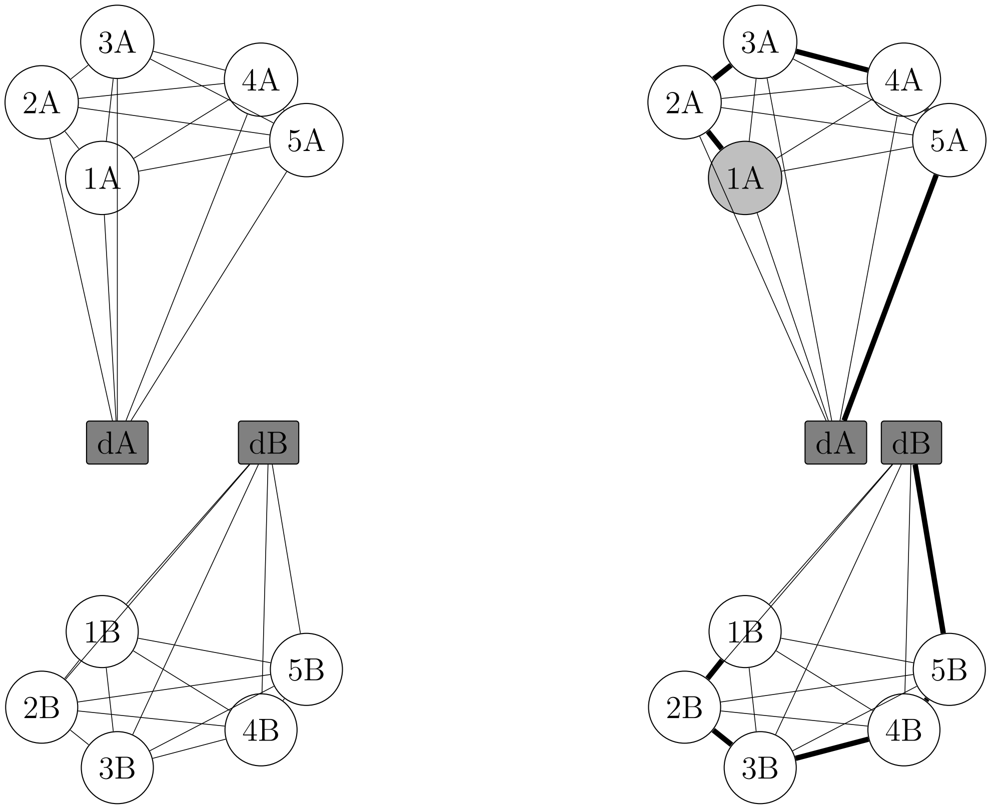
By leaving the free schedules as in this instance 3.17, we prove that an instance of the problem is NP-hard.

Return management

Take the instance of the Figure 3.17, we can imagine that the solution of return can be the same solution as to the opposite way, which proves that the complexity remains unchanged.

Different return locations

Let's take an example where the return is totally different.



(a) Vertices of the instance

(b) A solution

Figure 3.18: CPP with different return locations

We can break down the problem into two sub-problems corresponding both to the Hamiltonian Path Problem. So the complexity remains NP-Hard.

— 4 —
Problem solving model

In this chapter we will see our choices of resolution, and how we have implemented these choices.

4.1 Resolution specificities

Here is a description and a comparison of how to solve the problem with different models.

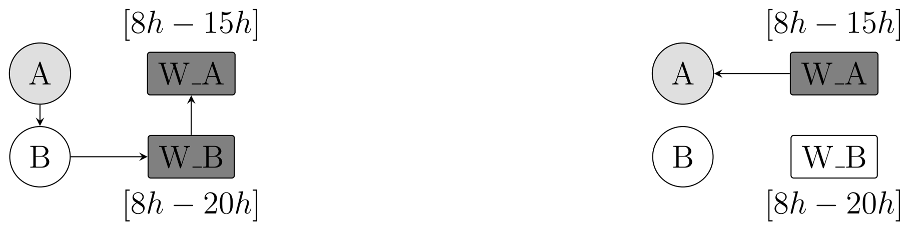
4.1.1 Different models

We have two types of models to implement in order to compare them:
Management of the return trip **associated** with the management of the home-to-work trip.
Management of the return trip **dissociated** with the management of the home-to-work trip.

In other words, **dissociate** means that we have a model that manages the home-to-work trip, and another that manages the return, based on the outcome of the go, ie the results of the program for the go are part of the program data for the return. **Associate** means that our program manages all the trips directly with the starting dataset.

For each of these two models, we have the case where the carpoolers are in the **same pool** for the home-to-work trip and the return trip, and the case where they can be in **different pools**

Let's see some examples in the following content:



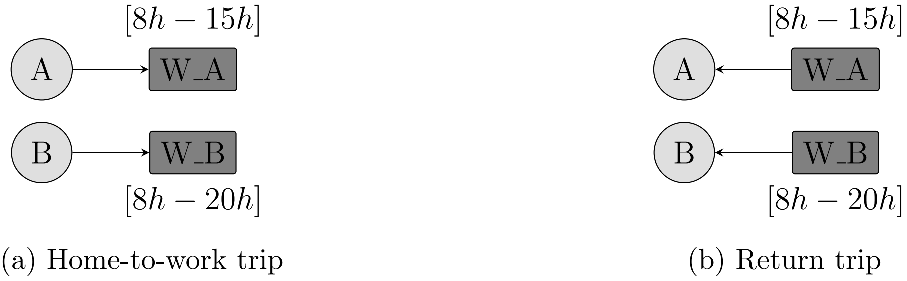
(a) Home-to-work trip

(b) Return trip

Figure 4.1: Dissociated programs

Figure 4.1 represents two implemented dissociated model, where one is executed for the home-to-work trip, and the other one for the corresponding return trip.
We introduce the notion of people who are **left behind**. We can observe it in (b), only taking

into account the schedules of work's beginning, this is obvious that we take only one car to save some fuel. However at the return trip, there are five hours of difference in schedules when they end the work. The person who took his car can go back home whereas the person who has not taken his car is stuck at work.
Of course, someone else would be able to pick up this person on the return run if there were more requests, but if this is not the case, this person is in the **left behind group**.
During the execution of these two dissociated models, there will be the number of people **left behind** in the solution.

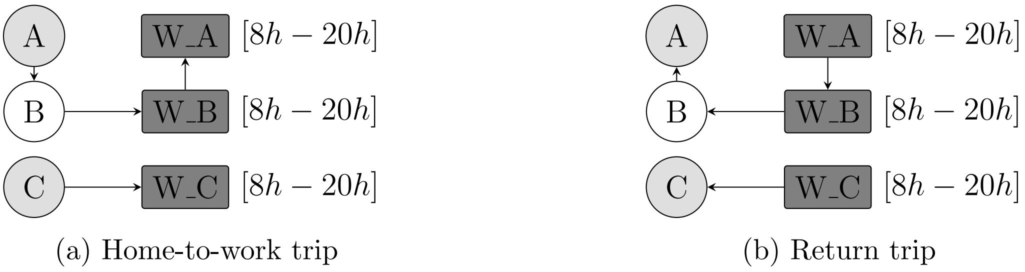


(a) Home-to-work trip (b) Return trip

Figure 4.2: Associated programs

Figure 4.2 shows that with the same initial data as in the Figure 4.1, we have a different solution. Indeed, this time the management of the work and return to work are done at the same time, in the same system of decision support. So the solution for this one, knowing that on the return they will need theirs cars, is to take each their car at the start.

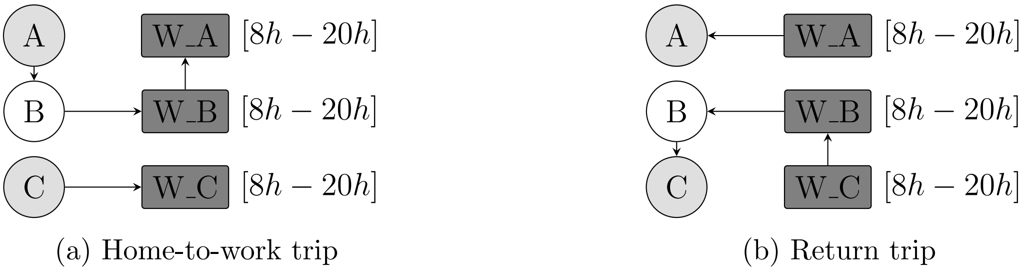
The usefulness of having both types of models makes it possible to compare the execution times on the same number of users, as well as the value of solutions.



(a) Home-to-work trip (b) Return trip

Figure 4.3: Same pools

In Figure 4.3 we can see that the system provides a solution to have the same pool of people in the home-to-work trip and in the return trip.



(a) Home-to-work trip (b) Return trip

Figure 4.4: Different pools

Figure 4.4 shows that leaving pools open, ie they can be made different during the both trips, constraining less and leaving more solutions, by abandoning the stability emitted in the Figure 4.3.

4.1.2 Heuristics used

Finding satellites

During the study, we want to see the impact of the use of gathering places. Indeed, the places that we will call Satellites, can represent exits of motorway where to deposit people, to facilitate the movement of the users drivers.
The use of these satellites is done in data pre-management, where for example, people can be dropped on a motorway exit within 1 km of their work.
This use of satellite points makes it possible to have fewer destinations, since they will be, for some, gathered in these satellites.

Chose the data

Grenoble is surrounded by mountain ranges, a heuristic to gather the following data from where people come from works perfectly, and is an optimal solution since people would not have interest to go around the mountain ranges.

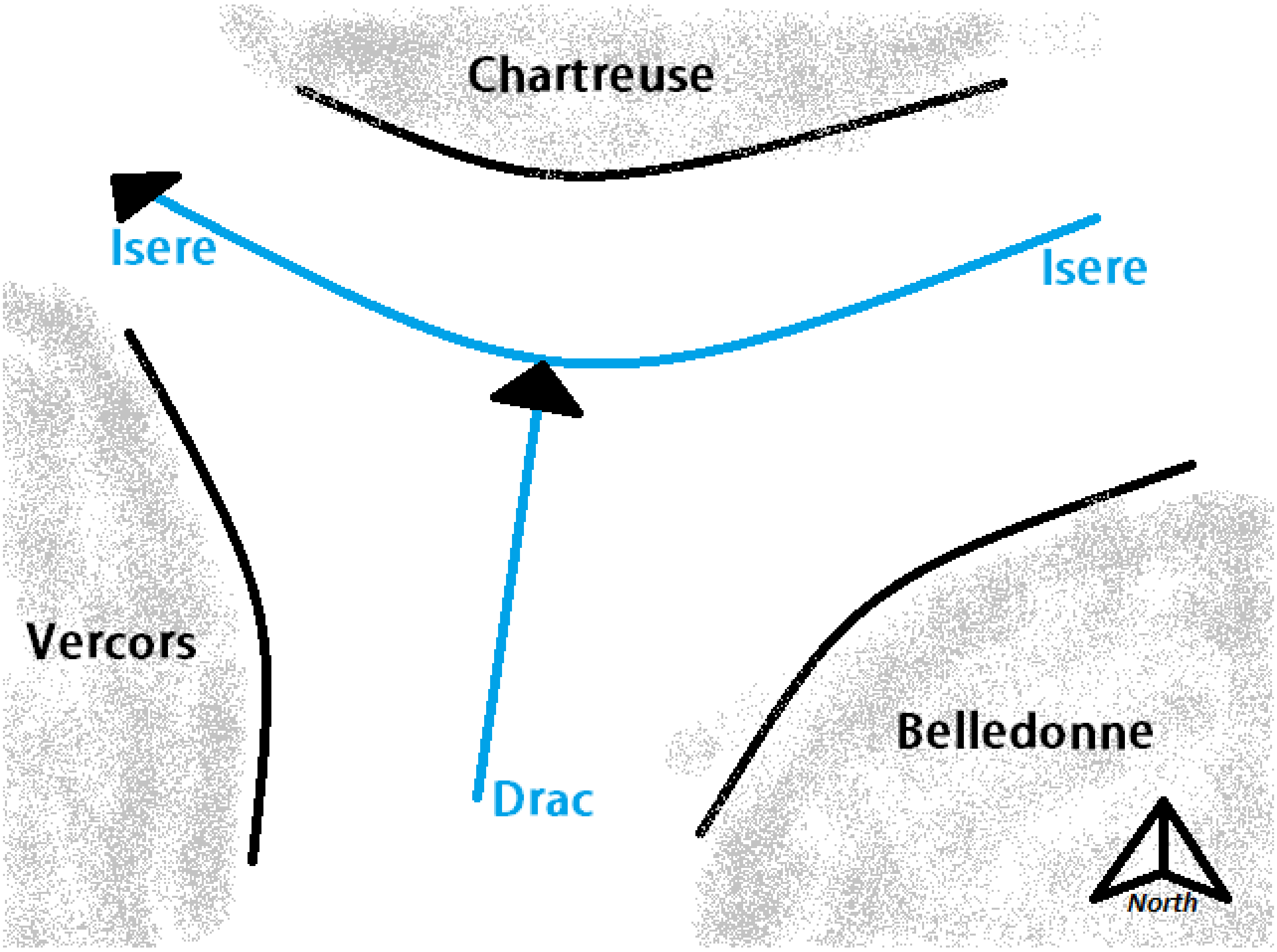


Figure 4.5: Mountains around Grenoble city

In Figure 4.5 the mountains are grayed and the blue arrows are rivers. This city configuration leaves only three major entrances, which are west, east and south. We can therefore create a group from each cardinality in order to increase the number of users taken into account by the decision system.

We will now see which data are used by our models.

4.2 The data model

In this section, we will see the data are used by our models.

4.2.1 A representation in graph form

The requests and the travels can be represented by a directed graph $G = (V, A)$ where $V = \{1, ..., 4n\}$ is the set of Vertices and $A = \{arc(i, j)\forall i, j \in V\}$ is the set of Arcs.
All destinations of a person are shifted by n from their first place.
For example the first user has the vertex pair $(1, 1 + n)$ corresponding to his home-work trip vertices, and the vertex pair $(1 + 2n, 1 + 3n)$ corresponding to his return trip.

4.2.2 Sets range

We have n users, each one having four summits, two for the home-work trip and two for the return trip respectively.

$O = \{1, ..., n\}$	Set of every possible driver/passenger.
$D = \{n + 1, ..., 2n\}$	Set of the workplaces.
$U = O \cup D$	Set of home-to-work places.
$K = \{2n + 1, ..., 3n\}$	Set of the origins to return (may be the workplace).
$L = \{3n + 1, ..., 4n\}$	Set of the destinations to return (may be home).
$W = K \cup L$	Set of return places.

4.2.3 Data used

In the dataset we have C the cost matrix for moving from point i to point j represents either a distance, or a cost of gas, or a duration. B and E corresponds to the start and end time of the work, and Q the capacity of the cars.

C_{ij}	$\forall i, j \in V$	Cost matrices for commuting $arc(i, j)$.
Q_n	$\forall n \in O$	Car's capacity of the user n.
B_v	$\forall v \in D$	Hour: Beginning of work for the user n.
E_v	$\forall v \in K$	Hour: End of work for the user n.
γ		Percentage of the initial travel time added to the max travel time.
δ		Constant value added to the max travel time.
α		Allowed waiting time to get to work early.
β		Allowed waiting time to leave work.
M		A large enough constant.

4.2.4 Diagram representation of data

In the following, the UML representation allows to visualize the data in another way.

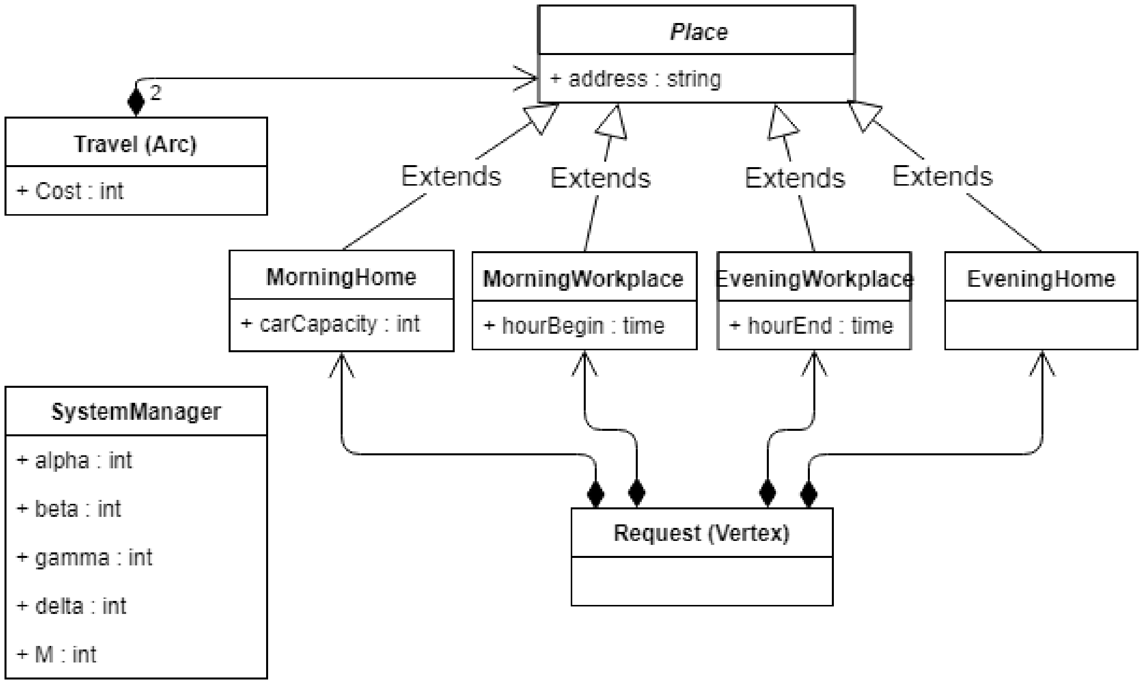


Figure 4.6: UML problem diagram

Figure 4.6 represents the data model.
The abstract class **Place** corresponds to the location of a place. Four classes inherit it, namely the place **MorningHome** that contains the capacity of the user's car, the place **MorningWorkplace** containing the time at which he starts work, the place **EveningWorkplace** containing the time at which his work ends, and the place **EveningHome** where he returns after work. Of course the workplaces can be the same, as well as the residential places. The **Travel(Arc)** class represents the arcs of the graph. This class takes two places to determine the time/distance between them. The **Request(Vertex)** class represents the vertices of the graph. This class takes the four places of the user's day in order to form a request composed of four vertices. The **SystemManager** class corresponds to what the person using the system must set for the mathematical model to work properly.

4.2.5 The dissociated models

The dissociated models have a first model taking the morning places, and a second model taking the evening places plus the result of those who take their cars in the first model.

The next section discusses the mathematical model of the Mixed Integer linear Programs we have implemented to solve the problem.

4.3 Mathematical modeling

We have binary variables representing the choices that must be made by the program, and variables representing sequencing, ie the time at which a driver would take a passenger, or the number of passengers inside. of this car after picking up or dropping people.

4.3.1 Decision variables

x_{ij}^k	$\forall k \in O, \forall i, j \in V$	$= 1$ if the travel of the $arc(i, j)$ is deserved by the driver k.
y^k	$\forall k \in O$	$= 1$ if the user k is a driver.
z_v^k	$\forall k \in O, \forall v \in V$	$= 1$ if the user k is picking or delivering the vertex v.
b_v^k	$\forall k \in O, \forall v \in V$	Estimated passage time of the driver k to the vertex v.
q_v^k	$\forall k \in O, \forall v \in V$	Number of persons in the car k at the vertex v.

We will now see what objectives our decision support system has.

4.4 Objective functions

Minimize

$\sum_{k \in O} \sum_{i,j \in V} C_{ij} \times x_{ij}^k$ The cost of all travels done

Finally we will see the general and specific constraints of our models. For the sequencing constraints used in our models, we were inspired by Mr. Karimi [KA15], then we discovered a version of the more recent formula written by Mr. Karoaglan [Kar18].

4.4.1 Constraints

Path constraints

Each vertex haven't more than one outgoing arc.

$$\sum_{k \in O} \sum_{j \in V, j \neq o} x_{oj}^k \leq 1 \qquad \forall o \in V$$

Each vertex haven't more than one ingoing arc.

$$\sum_{k \in O} \sum_{i \in V, i \neq d} x_{id}^k \leq 1 \qquad \forall d \in V$$

A driver picks-up himself before picking up someone else, so he has two ingoing arc.

$$\sum_{j \in V} x_{ij}^k \leq 2 \qquad \forall k \in O, \forall i \in V$$

A passenger has only one destination after being picked-up.

$$\sum_{j \in V, j \neq k, j \neq k+2n} x_{ij}^k \leq 1 \qquad \forall k \in O, \forall i \in V$$

A driver is picking himself.

$$y^k = z_k^k \qquad \forall k \in O$$

$$y^k = x_{kk}^k \qquad \forall k \in O$$

$$x_{kk}^k = z_k^k \qquad \forall k \in O$$

A non-driver can't pick himself.

$$x_{vv}^k = 0 \qquad \forall k \in O, \forall v \in V, v \neq k, v \neq (k+2n)$$

A vertex v is taken at most by one and only one driver.

$$\sum_{k \in O} z_v^k \leq 1 \qquad \forall v \in V$$

Prohibited cycles.

$$x_{ij}^k + x_{ji}^k \leq 1 \qquad \forall k \in O, \forall i \in V, \forall j \in U, i \neq j$$

Each driver must reach their destination.

$$\sum_{k \in O} \sum_{i \in V} x_{id}^k = 1 \qquad \forall d \in D \cup L$$

Each driver must leave their home and their workplace.

$$\sum_{k \in O} \sum_{j \in V, j \neq o} x_{oj}^k = 1 \qquad \forall o \in O$$

If k is picking someone he delivers him at destination.

$$z_o^k == z_{o+n}^k \qquad \forall k \in O, o \in O \cup K$$

A driver never goes away from his destination.

$$\sum_{v \in D \cup v \in L} x_{d,v}^k = 0 \qquad \forall k \in O \forall d = k+n \cup k+3n$$

If v is picked by k then k goes exactly one time to v.

$$z_v^k = \sum_{j \in U \cup W} x_{vj}^k \qquad \forall k \in O, \forall v \in U \cup W$$

Time constraints

The sum of the travels of a driver $\leq$ maximum time allowed on the road for this user.

$$\sum_{i,j \in U \cup W} x_{ij}^k \times C_{ij} \leq (1+\gamma\%) \times C_{k,k+n} + \delta \qquad \forall k \in O$$

Sequencing the hours of passages.

$$b_j^k \geq b_i^k - B_{k+n} + (C_{ij} + B_{k+n}) \times x_{ij}^k + (B_{k+n} - C_{ij}) \times x_{ji}^k \qquad \forall k \in O, \forall i,j \in U, i \neq j$$

$$b_j^k \geq b_i^k - M + (C_{i-2n,j-2n} + M) \times x_{ij}^k + (M - C_{i-2n,j-2n}) \times x_{ji}^k \qquad \forall k \in O, \forall i,j \in W, i \neq j$$

Everyone arrives between $[B - \alpha, B]$.

$$\sum_{k \in O} b_{v+n}^k \leq B_{v+n} \qquad \forall v \in O$$

$$\sum_{k \in O} b_{v+n}^k \geq B_{v+n} - \alpha \qquad \forall v \in O$$

Everyone leaves between $[arrivalTime - MaximalTravelTime, arrivalTime - C_{ij}]$.

$$\sum_{k \in O} b_v^k \leq (\sum_{k \in O} b_{v+n}^k) - C_{v,v+n} \qquad \forall v \in O$$

$$\sum_{k \in O} b_v^k \geq (\sum_{k \in O} b_{v+n}^k) - ((1 + \gamma\%) \times C_{k,k+n} + \delta) \qquad \forall v \in O$$

Passage time is zero if vertex is not chosen.

$$b_s^k \leq z_v^k \times M \qquad \forall k \in O, \forall v \in V$$

Capacity constraints

Each passenger can be in a car.

$$\sum_{k \in O} y^k \times Q_k \geq n$$

A driver k is his first passenger.

$$q_k^k = y^k \qquad \forall k \in O$$

Sequencing the number of passengers after visiting a vertex.

$$q_k^j \geq q_k^i - Q^k + (nPassengers_j + Q^k) \times x_{ij}^k + (Q^k - nPassengers_i) \times x_{ji}^k$$

$$\forall k \in O, \forall i \in U \cup W, \forall j \in U \cup W$$

If a driver k visits the vertex i then his capacity after visiting it can't exceed the capacity of his car.

$$q_k^v \leq Q^k \times z_v^k \qquad \forall k \in O, \forall v \in U \cup W$$

Return to home additional path constraints.
Each driver takes himself at the return.

$$x_{kk}^{k-2n} = y^{k-2n} \qquad \forall k \in K$$

Same pool management

In order to have the same pool on the way back, we can force users picked-up from home-to-work trip to be picked-up by the same driver to the return trip.

$$z_i^k = k_{i+2n} \qquad \forall k \in O, \forall i \in U$$

4.4.2 The use of the model

This mathematical model responds well to all the objectives that we set ourselves.

Indeed, the places of the return trip may be different than home-to-work trip ones, and the hours of work are respected.

We have access to the time values and capacity values at each vertex.

Limitations of the model

The number of users can not exceed thirty users because of the execution time which exponentially increases. The main cause is the management of sequencing, but it is necessary to not split it.

We will now see how the experiments occurred.

4.5 Resolution process

We have two execution processes for experiments. We define for each one a maximum execution time x of the linear program, as well as a number of run to make y.
The first presented in Figure 4.7 shows the case where we keep the same instance and vary parameters of the linear program.
The second presented in the Figure 1 shows a parameter variation at the problem level. It is therefore necessary to renew the instance of the problem with these new parameters in order to take them into account.

We will now see how these experiments unfolded.

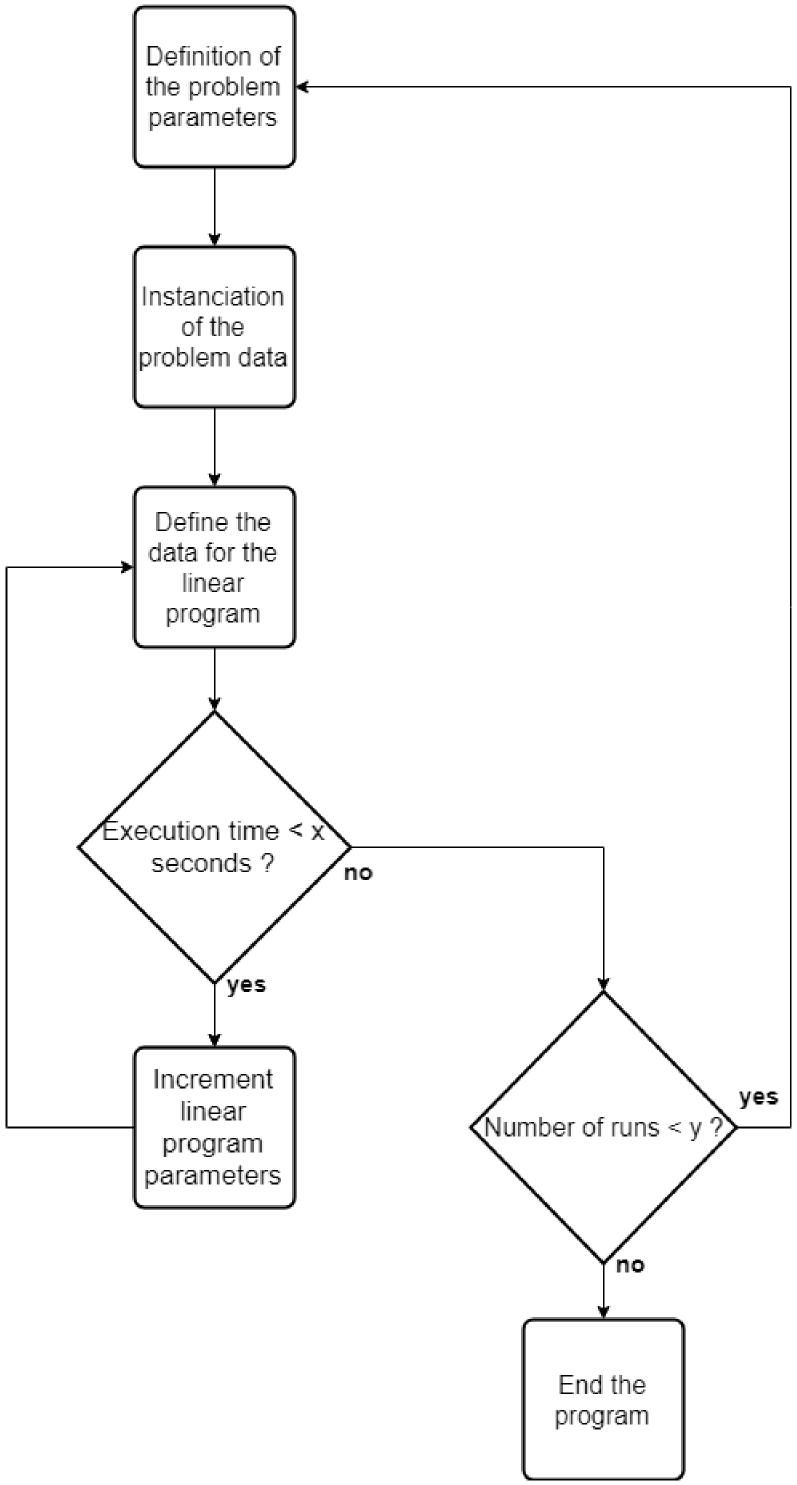

Definition of the problem parameters
Instanciation of the problem data
Define the data for the linear program
Execution time < x seconds ?
no
yes
Increment linear program parameters
Number of runs < y ?
yes
no
End the program

Figure 4.7: Flowchart of the LP's parameters variations

— 5 —
Experiments

We will see in each case which data we used, the value of the parameters, the results then we will analyze these results.

5.1 Experimental protocol

For all these experiments, we used the **CPLEX solver**. Our experimental platform is a computer with **8GB RAM memory**, an **Intel Core i5-4690 CPU 3.50 GHz**, and the operating system is **Windows 10 Professional 64-bit version**. The generator code and the solver code were written in **JAVA language**, and are available at the following address: `https://github.com/NeoKa4ra/CarPoolingInternship`.

5.2 About configurations

Here we will see how to understand the numbers given in the configurations. The JAVA lines of parameterization code being very explicit once understood, we will use them as configuration data throughout the experiments. In order to understand how this setting code works, we will detail each line of an example. **The car capacity of each user is always randomly between 2 and 4 seats**.

int nUsers = 5;
The number of users.

Before starting the next lines, it is necessary to define what is an experiment, a run and an execution. An experiment, called a run, is composed of several executions. The experience is therefore repeated over a number of run.

// Usage:(modeLP, modeInstance, numberofRun, execInstanceTimeMax, execTotalTimeMax, nMaxUsers, suffix)

GS = new GlobalSettings(Constants.GLPWR, Constants.GDI, 10, 300, 6, 12, ”varyUsersWR”);

Here we have the parameters for the selected linear program, the instance mode, i.e. whether it is renewed between each execution or not, the number of runs of the experiment, the maximum time of an execution that stops the program if it is exceeded, the maximum total time that stops the execution if it is exceeded, the maximum number of users that stops the execution if it is reached, and a callsign for the test and result files.

We enter the two parameters of the instance, those concerning the time and those concerning the matrix. They have been separated to have fewer parameters on a single line, to improve understanding.

// (matrixMode, nbPersons, matrixRange, citiesList, workplacesList, probScdWork, probScdHome)
MS = new MatriceSettings(Constants.MCPWP, nUsers, 200, cities, workplaces, 20, 5);
We have the matrix mode we want to have (completely random, etc.), the number of users, the range of each random instance of the matrix, the list of city coordinates, then work location coordinates, the probability of having a second job and the probability of having a second home.

// Usage:(nPersons, morningHour, morningHourRange, eveningHour, eveningHourRange)
HS = new HoursSettings(nUsers, 850, 50, 1500, 500);
We have the number of users, the interval in which the working time must be in the morning, then the interval in which the working time must be in the evening. Note that the hours are in percentage of hours for simplicity's sake, 50 units corresponding to 50% of one hour, i.e. 30 minutes.

// Usage:LPSettings(advance, waitingTime, deviationPercentage, deviationValue)
LPS = new LPSettings(50, 25, 20, 20);
We have the maximum advance time to get to work, the longest waiting time after work, the percentage of deviation allowed and the fixed value of deviation allowed.

// (varyNUsers, varyAdvance, varyWaitingTime, varyDeviationPercentage, varyDeviationValue)
LPVS = new LPVariationsSettings(1, 0, 0, 0, 0);
Here we have all possible variations for the parameters of the linear program, each number corresponds to the incrementation made between each execution.

5.3 Variations in LP parameters & users

In this section, we will see how the objective and the execution time behave according to the variation of the linear program parameters.

5.3.1 Benchmarks

For the first tests, we used a distance matrix filled **randomly** in a range from 0 to 200, we will call it ”All random” when it is filled completely randomly and ”Random close homes” when the random is oriented so that people are closer to each other, as well as workplaces.

5.3.2 Configuration

Here are the parameters used in the following experiments.

int nUsers = x;
The number of users.

// Usage:(modeLP, modeInstance, numberofRun, execInstanceTimeMax, execTotalTimeMax, nMaxUsers, suffix)
GS = new GlobalSettings(Constants.GLPWR, Constants.GSI, y, 300, 6, 25, "");
The execution was done on the **linear program including the return path**, with the same instance at each run. The execution time was **300 seconds** maximum and the **total time was 6 minutes**. We had either **10 runs per experiment or 20**.

// (matrixMode, nbPersons, matrixRange, citiesList, workplacesList, probScdWork, probScdHome)
MS = new MatriceSettings(Constants.MM, nUsers, 200, cities, workplaces, 0, 0);
The data were generated randomly, refer to benchmark.

// Usage:(nPersons, morningHour, morningHourRange, eveningHour, eveningHourRange)
HS = new HoursSettings(nUsers, 850, 50, 1500, 500);
Working hours were between **8:30 and 9:00 in the morning** and between **15:00 and 20:00 in the evening**.

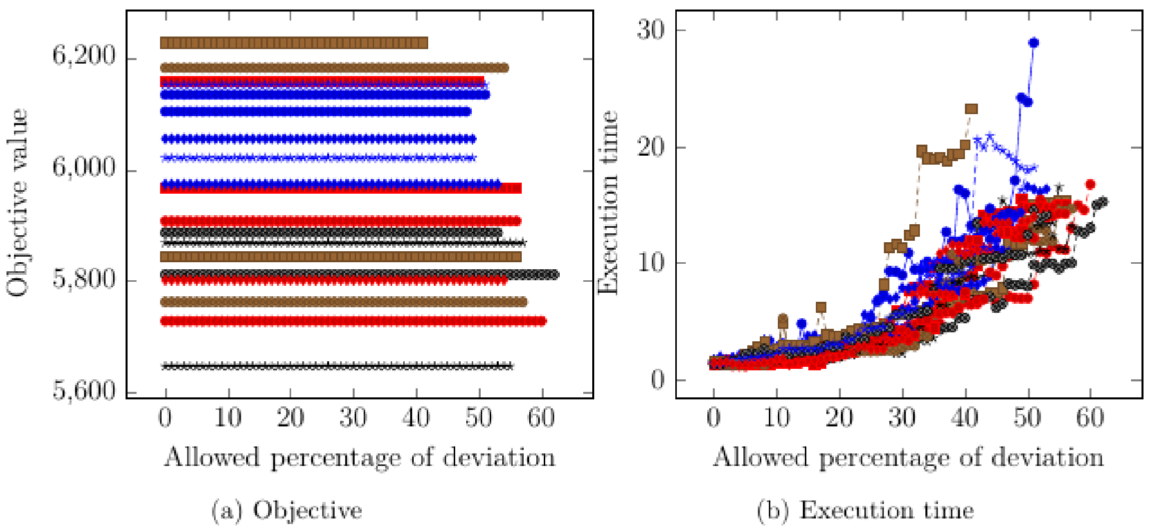
// Usage:LPSettings(advance, waitingTime, deviationPercentage, deviationValue)
LPS = new LPSettings(50, 25, 20, 20);
When one parameter varied, the others were always set to the base value indicated above.

// (varyNUsers, varyAdvance, varyWaitingTime, varyDeviationPercentage, varyDeviationValue)
LPVS = new LPVariationsSettings(0, 1, 1, 1, 1);
When a parameter varies, it is first set to zero and then incremented at each execution. Here we have all possible variations for the parameters of the linear program, each number corresponds to the incrementation made between each execution. We will see the variation on these parameters.

5.3.3 Results

As all the behaviors were similar, we studied one and put the results of this experiment in the appendix, see 5.7.2.

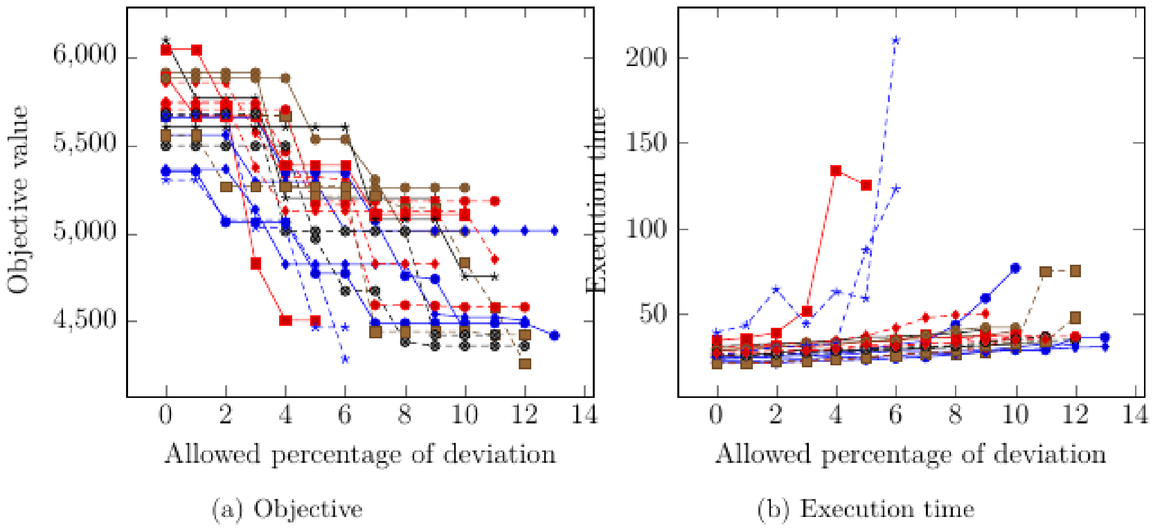
Varying the deviation's percentage of the travel allowed



(a) Objective (b) Execution time

Figure 5.1: Vary the deviation percentage with all random and with 20 users

In the Figure 5.1 we can see that the objective values remain the same regardless of the percentage deviation allowed, up to a maximum of 50% . The execution time increases with the percentage of deviation.



(a) Objective (b) Execution time

Figure 5.2: Vary the deviation percentage with random close homes and with 19 users

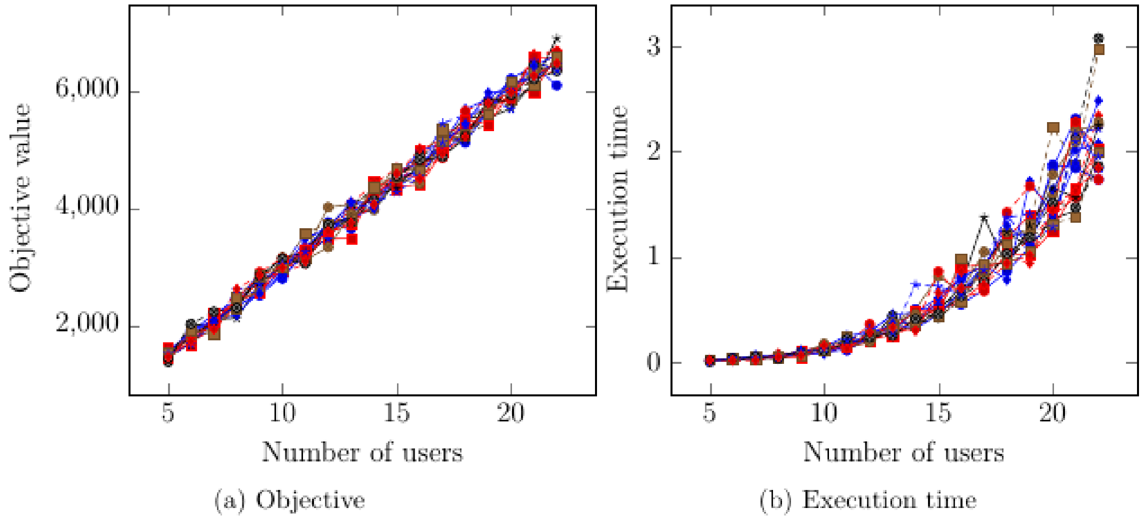
In the Figure 5.2, we can see that the objective value decreases with the percentage deviation while the execution time tends to increase.

5.3.4 Analysis

The test for all random values Figure 5.1 cannot be reduced with such a small percentage deviation, given the impressive gap that can be made between houses and between workplaces. The

corresponding execution time seems to increase exponentially with the percentage deviation, this is explained by more relaxed constraints, leaving a larger decision space.
However, when we bring homes and workplaces closer together ref: Figure 5.2, we have an objective value that decreases with the percentage of deviation, which is explained because people living close and their workplaces being close, if we increase the possible travel time of each user he is more able to take his neighbor. For some instances, the execution time explodes much faster than for all random values. This is explained by a much larger decision space caused by the more numerous possibilities, left by the fact that people can much more easily take someone or someone else in view of the configuration.

Varying the number of users



(a) Objective

(b) Execution time

Figure 5.3: Vary users all random

In Figure 5.3, we can observe a linear increase of the objective value with the number of users while the execution time seems to grow exponentially.

5.3.5 Analysis

The linear increase in the objective value is explained by the increasing number of users, since each user is taken into account in the calculation of the objective function. Since we have proven that this problem is NP-hard, we have an execution time that increases exponentially with the number of users.

5.4 Associated/dissociated programs

We test and compare the associated linear program with the first part of the dissociated linear program.

5.4.1 Benchmarks

We use our JAVA coded generator, with randomly generated cities. These cities are actually condensed points representing users. Moreover, we have fixed everyone's work at one and the same point.

5.4.2 Configuration

We wish to observe the difference in execution time of the two types of programs.

int nUsers = x;
The number of users.

// Usage:(modeLP, modeInstance, numberofRun, execInstanceTimeMax, execTotalTimeMax, nMaxUsers, suffix)
GS = new GlobalSettings(Constants.GLPWR, Constants.GDI, 30, 300, 6, 12, "");
The mode is presented in the Benchmarks, and we have a **different instance generated each time**, since we vary the number of people. We did **30 runs** per experiment.

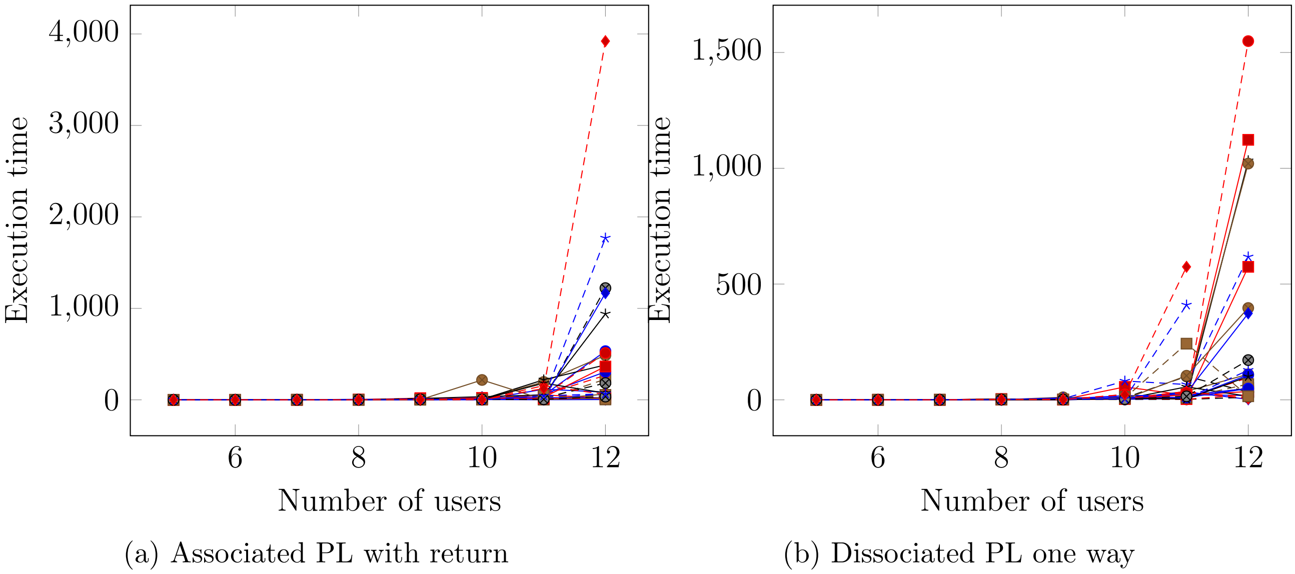
// (matrixMode, nbPersons, matrixRange, citiesList, workplacesList, probScdWork, probScdHome)
MS = new MatriceSettings(Constants.MCPWP, nUsers, 200, cities, workplaces, 20, 5);
The mode is presented in the Benchmarks. We put **20% chance** that a person would have a **different workplace** on return, and **5% chance** that he would have a **different destination** on return.

// Usage:(nPersons, morningHour, morningHourRange, eveningHour, eveningHourRange)
HS = new HoursSettings(nUsers, 900, 100, 1500, 100);
To be fair, we put **1 hour apart** in the morning and evening, people arrive at work between 9:00 and 10:00 and leave between 15:00 and 16:00.

// Usage:LPSettings(advance, waitingTime, deviationPercentage, deviationValue)
LPS = new LPSettings(50, 25, 20, 5);
We think it's good to have a **maximum of 30 minutes of advance** to get to work and **15 minutes to wait** when you finish the job. Moreover, we estimate that **5+20% of the trip** is enough not to discourage people from carpooling.

// (varyNUsers, varyAdvance, varyWaitingTime, varyDeviationPercentage, varyDeviationValue)
LPVS = new LPVariationsSettings(1, 0, 0, 0, 0);
We vary the number of users.

5.4.3 Results



(a) Associated PL with return

(b) Dissociated PL one way

Figure 5.4: Comparison of the execution times of the dissociated/associated PLs

In Figure 5.4, we can see that both programs have an execution time exploding to 11 users.

5.4.4 Analysis

Dissociated programs seem to explode to the same number of users as the associated program. It does not justify enough deviation to be more useful than the associated program. In the following, we will only use the dissociated program.

5.5 Users have the same workplace from generated cities

We want to observe a solution given by our linear program when there are a few cities and a single workplace.

5.5.1 Benchmarks

We use our JAVA coded generator, with randomly generated cities. These cities are actually condensed points representing users. Moreover, we have fixed everyone's work at one and the same point.

5.5.2 Configuration

Here is the configuration used.

```
int nUsers = 10;
```

We took **10 users**.

// Usage:(modeLP, modeInstance, numberofRun, execInstanceTimeMax, execTotalTimeMax, nMaxUsers, suffix)

GS = new GlobalSettings(Constants.GLPWR, Constants.GSR, 1, 300, 6, 12, ””);

We are in **Single Run mode**, which means that we will only run the program once no matter what happens.

// (matrixMode, nbPersons, matrixRange, citiesList, workplacesList, probScdWork, probScdHome)

MS = new MatriceSettings(Constants.MCPWP, nUsers, 200, cities, workplaces,0,0);

The mode is presented in the Benchmarks. We have **3 cities**.

// Usage:(nPersons, morningHour, morningHourRange, eveningHour, eveningHourRange)

HS = new HoursSettings(nUsers, 800, 100, 1600, 100);

We left a beat of **an hour** for the beginnings and ends of work.

// Usage:LPSettings(advance, waitingTime, deviationPercentage, deviationValue)

LPS = new LPSettings(50, 25, 20, 5);

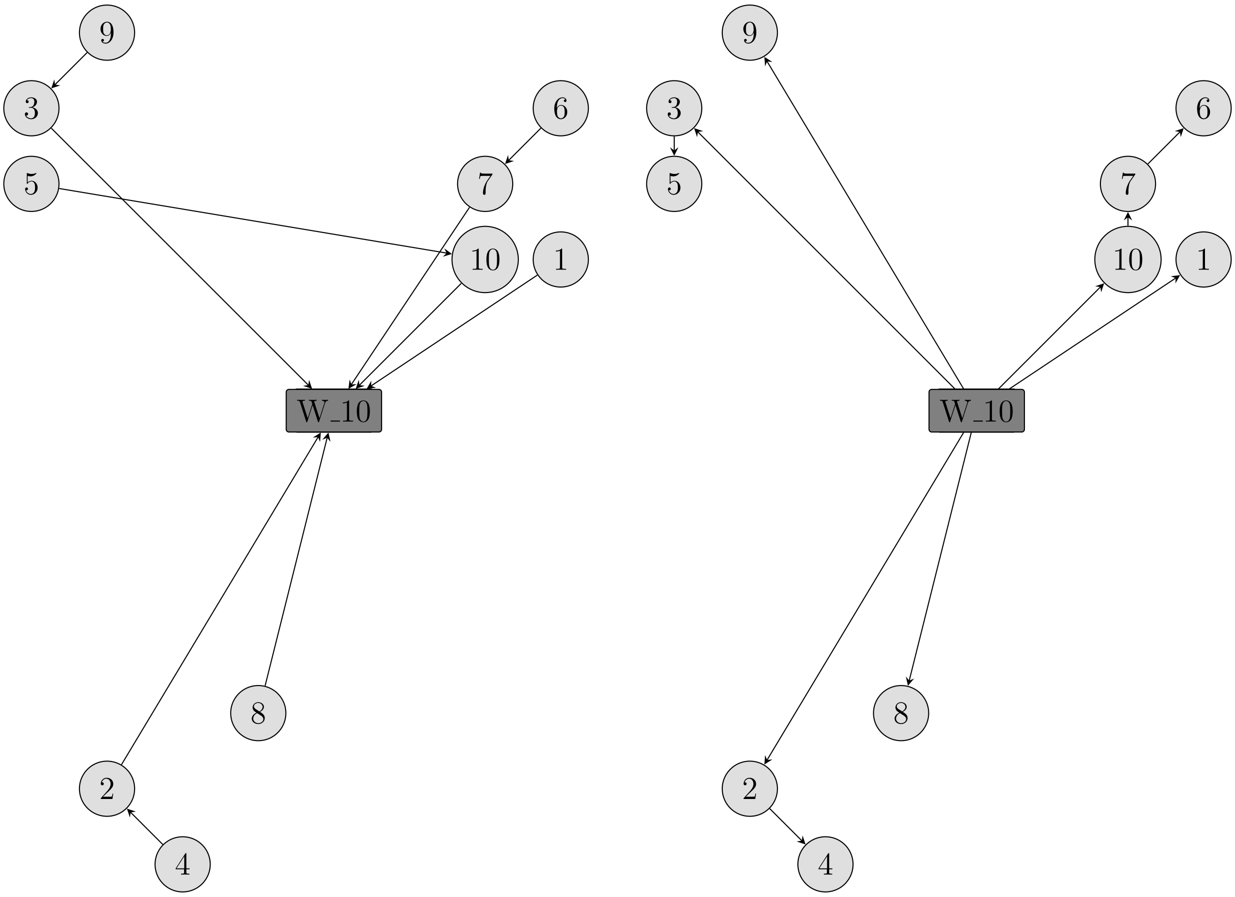
We think it’s good to have a **maximum of 30 minutes of advance** to get to work and **15 minutes to wait** when you finish the job. Moreover, we estimate that **5+20% of the trip** is enough not to discourage people from carpooling.

// (varyNUsers, varyAdvance, varyWaitingTime, varyDeviationPercentage, varyDeviationValue)

LPVS = new LPVariationsSettings(0, 0, 0, 0, 0);

We do not vary anything

5.5.3 Results



(a) Home-to-work trip

(b) Return trip

Figure 5.5: An execution with the same place of work

In Figure 5.5,we can see that cities are distinct. A user goes to pick someone up in another city on the way out, and everyone goes straight back to their city on the way back.

5.5.4 Analysis

The user who went to pick up someone in the other city allows to reduce the travelled distance since the solution is made with a car in less in the return trip, while respecting car capacity constraints and working hours.

5.6 Heuristics in the case of Grenoble

As we have seen, Grenoble is a city with only three entrances due to the mountains surrounding it. We used GoogleMapAPI to get the contact details of cities near Grenoble, which we scaled up to use within our program. We could not use GoogleMapAPI correctly because of the limited number of requests per day (5 per day)

5.6.1 Benchmarks

We use our JAVA coded generator, with **chosen cities**. Moreover, we have fixed everyone's work at one and the same point by speculating that workplaces are clustered, or that people are dropped off at highway exits.

5.6.2 Configuration

We wish to observe the difference in execution time of the two types of programs.

int nUsers = 20,18;
The number of users.

// Usage:(modeLP, modeInstance, numberofRun, execInstanceTimeMax, execTotalTimeMax, nMaxUsers, suffix)
GS = new GlobalSettings(Constants.GLPWR, Constants.GSR, 1, 300, 6, 12, "");
We are in **Single Run mode**, which means that we will only run the program once no matter what happens.

// (matrixMode, nbPersons, matrixRange, citiesList, workplacesList, probScdWork, probScdHome)
MS = new MatriceSettings(Constants.MCPWP, nUsers, 200, cities, workplaces,0,0);
The mode is presented in the Benchmarks. We have **3 cities**.

// Usage:(nPersons, morningHour, morningHourRange, eveningHour, eveningHourRange)
HS = new HoursSettings(nUsers, 800, 100, 1600, 500);
We left a beat of **an hour** for the beginnings and ends of work.

// Usage:LPSettings(advance, waitingTime, deviationPercentage, deviationValue)
LPS = new LPSettings(50, 25, 20, 5);
We think it's good to have a **maximum of 30 minutes of advance** to get to work and **15 minutes to wait** when you finish the job. Moreover, we estimate that **5+20% of the trip** is enough not to discourage people from carpooling.

// (varyNUsers, varyAdvance, varyWaitingTime, varyDeviationPercentage, varyDeviationValue)
LPVS = new LPVariationsSettings(0, 0, 0, 0, 0);
We do not vary anything.

The correspondence of points to cities is approximate because of the Java Point class which only accepts integers.
First simulation: new Point(-3, 5)); // VOIRON cities.add(new Point(-1, -4)); // VIF cities.add(new Point(4, 4)); // CROLLES

Second simulation: cities.add(new Point(-3, 5)); // VOIRON cities.add(new Point(-5, 1)); // VINAY cities.add(new Point(-2, 7)); // ST LAURENT DU PONT

Third simulation: cities.add(new Point(10, 10)); // PONTCHARRA cities.add(new Point(7, 7)); // LE TOUVET cities.add(new Point(4, 4)); // CROLLES

Fourth simulation: cities.add(new Point(10, 10)); // VIZILLE cities.add(new Point(7, 7)); // PONT DE CLAIX cities.add(new Point(4, 4)); // VIF

5.6.3 Results

For reasons of space the rest of the results are in appendix 5.7.2.

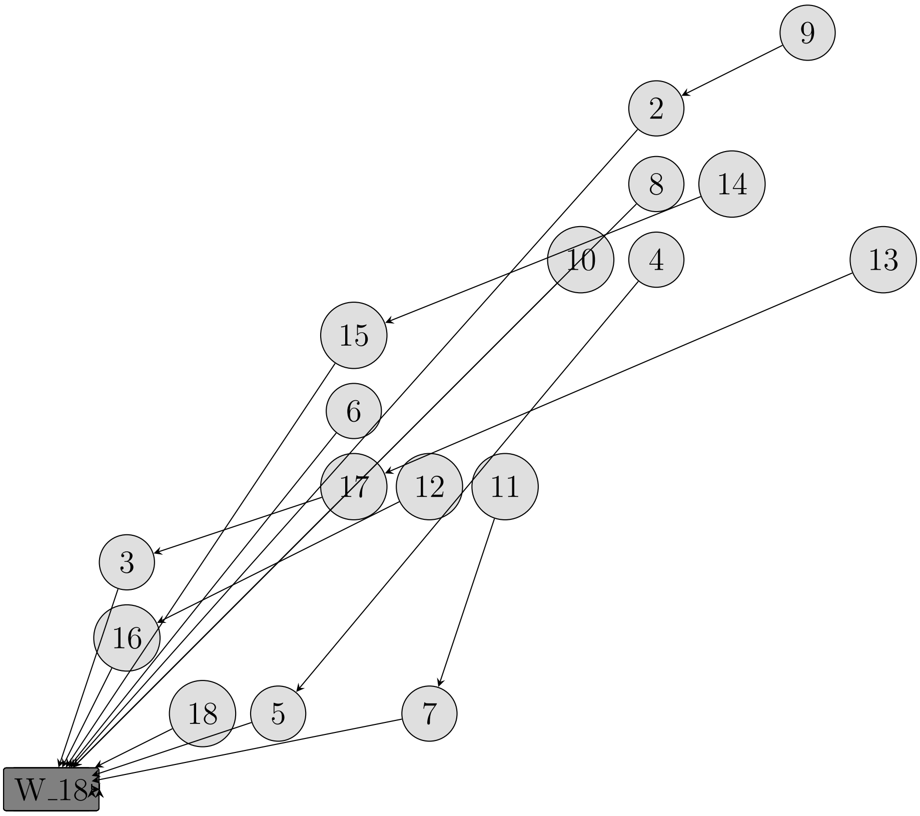


Figure 5.6: Home-to-work : PONTCHARRA LE-TOUVET CROLLES

In Figure 5.6 we observe that cities in this direction are aligned.

5.6.4 Analysis

In all these Figures, we can see that there are two distinct configurations, being the aligned and scattered city configurations. Heuristics in each direction being chosen as the best match could be implemented.

5.7 Limitations of the linear program

We have seen that we do not exceed 25 users with this linear program. Even with the heuristics of the Grenoble particularity, we would not exceed 75 users for an exact method.

5.7.1 About the fill rate of the cars

We studied with recent parameters, i.e. random cities with 5+20% deviation, 30 minutes advance and 15 minutes wait after work allowed, 20% chance of having a second place of work and 5% chance of having a second home. We have varied the number of users over 30 runs at each experiment, with a random number of work schedules first from 5:00 hours, then 1:00 hours and finally everyone at the same schedules.

Table 5.1: Vehicle fill rate (average number of people)

5:00	1:00	0:00
1.14	1.41	1.81
1.04	1.38	1.89
1.19	1.36	1.94
1.15	1.37	1.68
1.16	1.37	1.75

We observe that the fill rate needs many more users to have more possible matches in general so that it is increased significantly. Indeed, we can see that by bringing people's schedules closer together, the vehicle fill rate increases, and this without taking into account their location in relation to places of work.

5.7.2 What remains to be done

However, we could try to **divide the scheduling parts** of the program and observe the number of users that can be processed, and analyze the accuracy of the results. Moreover, our approximation of cities and routes is as the crow flies, but it should be known that the arrival to the northwest is like a funnel for cars, we should take into account **exact data**. Finally, we could **decompose each direction coming to Grenoble** as a heuristic. For example the heuristic **ant colony for alignment**, gather users by **work end time** for others.

Conclusion and perspectives

During this study, we saw that managing the return trip does not increase the complexity compared to the simple management of the outward trip, whereas it would allow users to have more flexibility, as well as multi-destination during the same day. We have also seen that the correspondence of schedules strongly allows the creation of user groups.

It remains to try to apply heuristics according to the origin of users, as well as the decomposition of the exact method to see if it is possible to increase the number of users without too much deteriorating the quality of solutions.

Bibliography

[ARN18] SYLVAIN ARNULF. Covoiturage domicile/travail : Klaxit (ex-Wayzup) embarque de nouveaux partenaires pour se detacher, 2018.

[BB04] Jean Berger and Mohamed Barkaoui. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 31(12):2037–2053, October 2004.

[BCCL11] Maurizio Bruglieri, Diego Ciccarelli, Alberto Colorni, and Alessandro Lue. PoliUniPool: a carpooling system for universities. *Procedia - Social and Behavioral Sciences*, 20:558–567, January 2011.

[BGG$^+$17] Michele Berlingerio, Bissan Ghaddar, Riccardo Guidotti, Alessandra Pascale, and Andrea Sassi. The GRAAL of carpooling: GReen And sociAL optimization from crowd-sourced data. *Transportation Research Part C: Emerging Technologies*, 80:20–36, July 2017.

[BH06] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, April 2006.

[BMM04] Roberto Baldacci, Vittorio Maniezzo, and Aristide Mingozzi. An Exact Method for the Car Pooling Problem Based on Lagrangean Column Generation. *Operations Research*, 52(3):422–439, June 2004.

[CdLHM04] Roberto Wolfler Calvo, Fabio de Luigi, Palle Haastrup, and Vittorio Maniezzo. A distributed geographic information system for the daily car pooling problem. *Computers & Operations Research*, 31(13):2263–2278, 2004.

[CL03] Jean-Franois Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[Cor06] Jean-Franois Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3):573–586, June 2006.

[CV09] G. Correia and J. M. Viegas. A conceptual model for carpooling systems simulation. *J Simulation*, 3(1):61–68, March 2009.

[DG16] Patricia Delhomme and Alexandra Gheorghiu. Comparing French carpoolers and non-carpoolers: Which factors contribute the most to carpooling? *Transportation Research Part D: Transport and Environment*, 42:1–15, January 2016.

[Dio] Daniel Dionne. BlaBlaCar monopolise le covoiturage : et la libre concurrence, alors ?

[GLS96] Michel Gendreau, Gilbert Laporte, and Rene Seguin. A tabu search heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research*, 44(3):469–477, 1996.

[Guo12] Yuhan Guo. *Metaheuristics for solving large size long-term car pooling problem and an extension.* Artois, November 2012.

[HHJL08] William Ho, George T. S. Ho, Ping Ji, and Henry C. W. Lau. A hybrid genetic algorithm for the multi-depot vehicle routing problem. *Engineering Applications of Artificial Intelligence*, 21(4):548–557, June 2008.

[HJL15] Shih-Chia Huang, Ming-Kai Jiau, and Chih-Hsiang Lin. A genetic-algorithm-based approach to solve carpool service problems in cloud computing. *IEEE Transactions on Intelligent Transportation Systems*, 16(1):352–364, 2015.

[HYB00] Hai-Jun Huang, Hai Yang, and Michael G. H. Bell. The models and economics of carpools. *Ann Reg Sci*, 34(1):55–68, March 2000.

[KA15] Ismail Karaoglan and Fulya Altiparmak. A memetic algorithm for the capacitated location-routing problem with mixed backhauls. *Computers & Operations Research*, 55:200–216, March 2015.

[Kar18] Hossein Karimi. The capacitated hub covering location-routing problem for simultaneous pickup and delivery systems. *Computers & Industrial Engineering*, 116:47–58, February 2018.

[Kot04] Amit B. Kothari. Genghis-a multiagent carpooling system. *Bath: Department of Computer Science, University of Bath*, 2004.

[LHCY08] J. Lau, W. T. Hung, C. S. Cheung, and D. Yuen. Contributions of roadside vehicle emissions to general air quality in Hong Kong. *Transportation Research Part D: Transport and Environment*, 13(1):19–26, January 2008.

[LL03] Haibing Li and Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(02):173–186, 2003.

[MCH04] Vittorio Maniezzo, Antonella Carbonaro, and Hanno Hildmann. An ANTS heuristic for the longterm car pooling problem. In *New optimization techniques in engineering*, pages 411–430. Springer, 2004.

[ML04] Silvia Mazzeo and Irene Loiseau. An Ant Colony Algorithm for the Capacitated Vehicle Routing. *Electronic Notes in Discrete Mathematics*, 18:181–186, December 2004.

[OB04] Godfrey C. Onwubolu and B. V. Babu. *New Optimization Techniques in Engineering.* Studies in Fuzziness and Soft Computing. Springer-Verlag, Berlin Heidelberg, 2004.

[PDH08] Sophie Parragh, Karl Doerner, and Richard Hartl. A survey on pickup and delivery problems: Part II: Transportation between pickup and delivery locations. *Journal fr Betriebswirtschaft*, 58:81–117, June 2008.

[PHDR04] Michael Polacek, Richard F. Hartl, Karl Doerner, and Marc Reimann. A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows. *J Heuristics*, 10(6):613–627, December 2004.

[Rus95] Robert A. Russell. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 29(2):156–166, May 1995.

[Sag74] James S. Sagner. The impact of the energy crisis on American cities based on dispersion of employment, utilization of transit, and car pooling. *Transportation Research*, 8(4):307–316, October 1974.

[Str07] Stephen Stradling. Determinants of Car Dependence Threats from Car Traffic to the Quality of Urban Life, 2007.

[VCG$^+$12] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[Web08] Web. Occupancy rates of passenger vehicles, 2008.

[Web16] Web. Karos, comment a marche ? - Le Parisien, 2016.

[Web17] Web. #Covoiturage domicile-travail BlaBlaLines arrive Paris et en Ile-de-France, 2017.

[Web18] Web. Covoiturage gratuit en ile-de-France pendant les jours de greve !, 2018.

[Woh70] Martin Wohl. A METHODOLOGY FOR FORECASTING PEAK AND OFF-PEAK TRAVEL VOLUMES. *Highway Research Record*, (322), 1970.

[YC11] Shangyao Yan and Chun-Ying Chen. An optimization model and a solution algorithm for the many-to-many car pooling problem. *Ann Oper Res*, 191(1):37–71, November 2011.

Appendix

Second resolution process

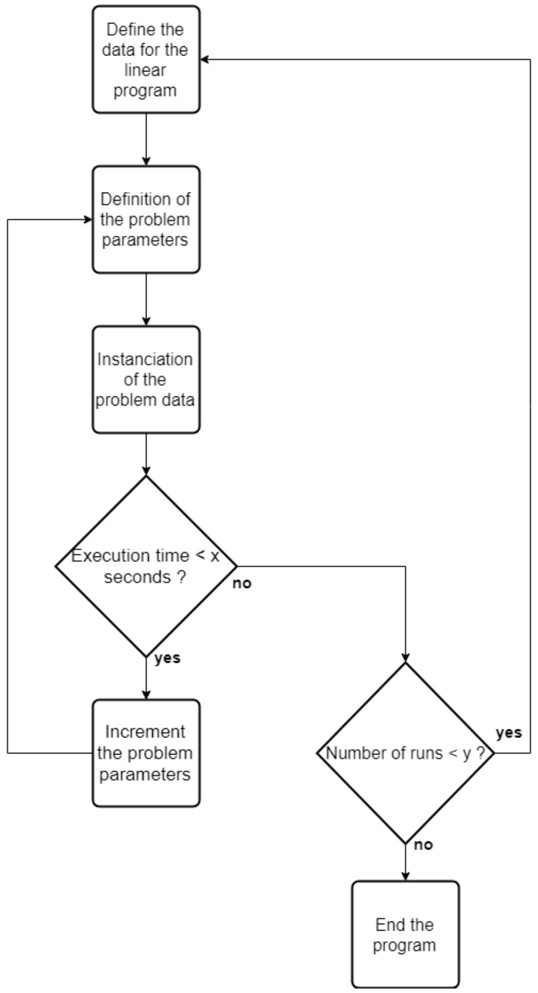
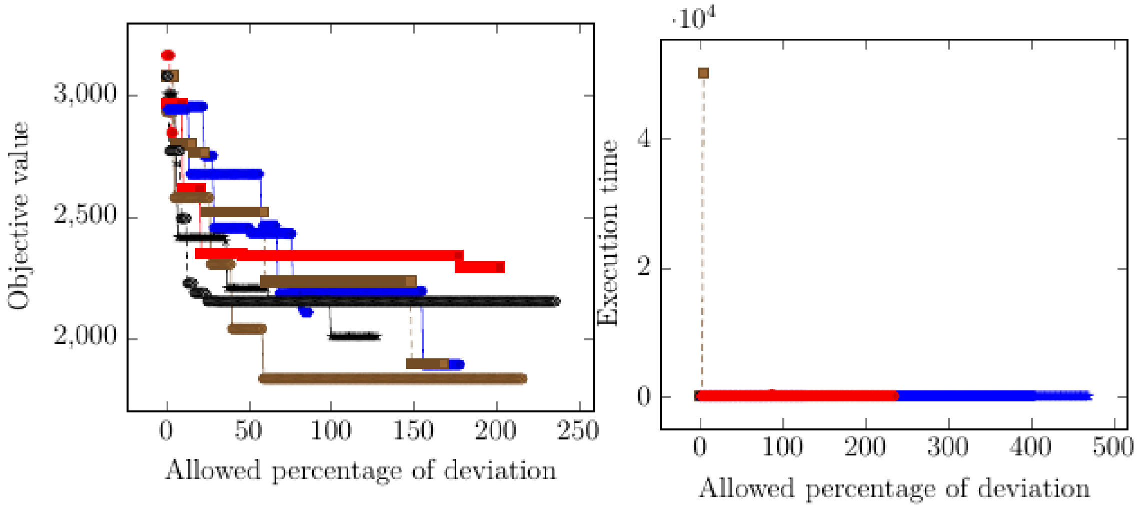

Define the data for the linear program
Definition of the problem parameters
Instanciation of the problem data
Execution time < x seconds ?
no
yes
Increment the problem parameters
Number of runs < y ?
yes
no
End the program

Figure 1: Flowchart of the problem's parameters variations

Continuation of the results of the LP parameter variations

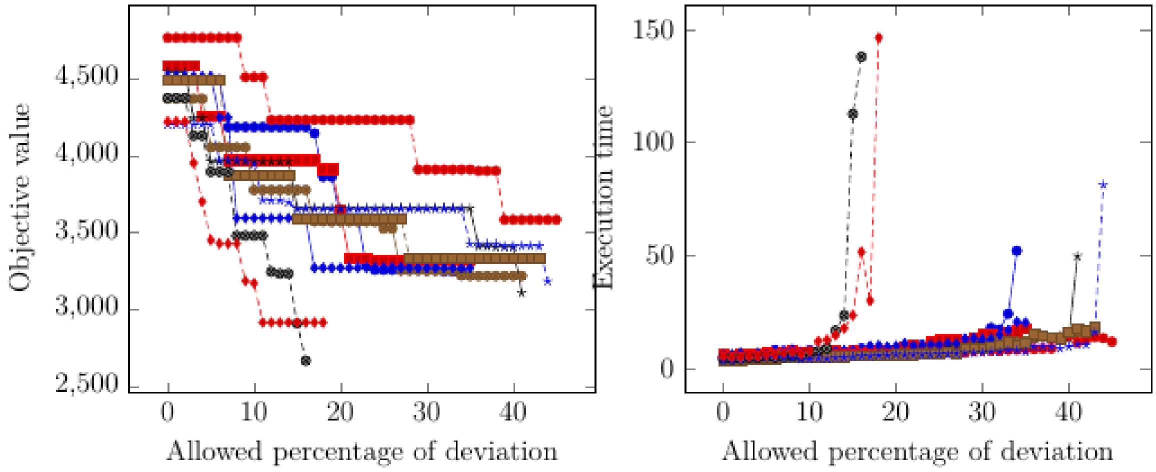
Varying the deviation's percentage of the travel allowed



(a) Objective

(b) Execution time

Figure 2: Vary the deviation percentage with random close homes and with 10 users

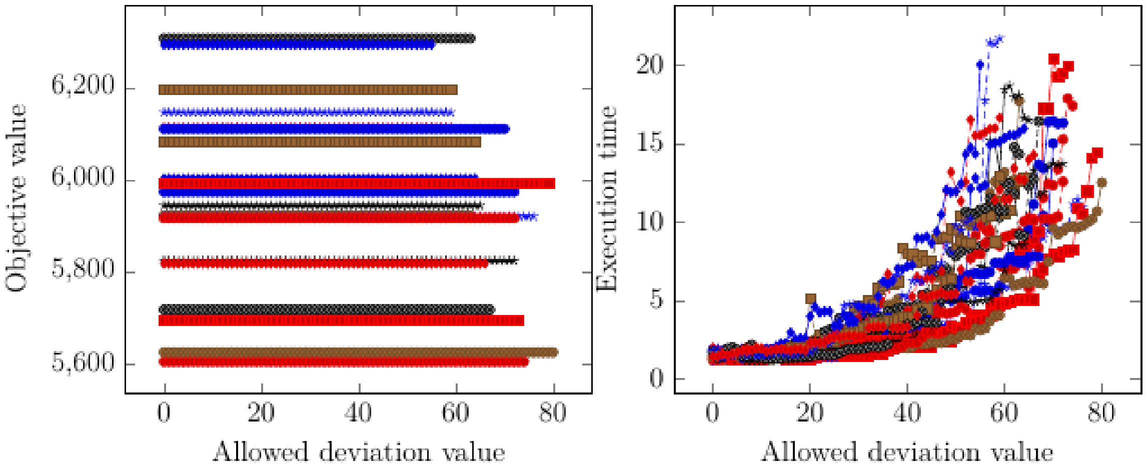


(a) Objective

(b) Execution time

Figure 3: Vary the deviation percentage with random close homes and with 15 users

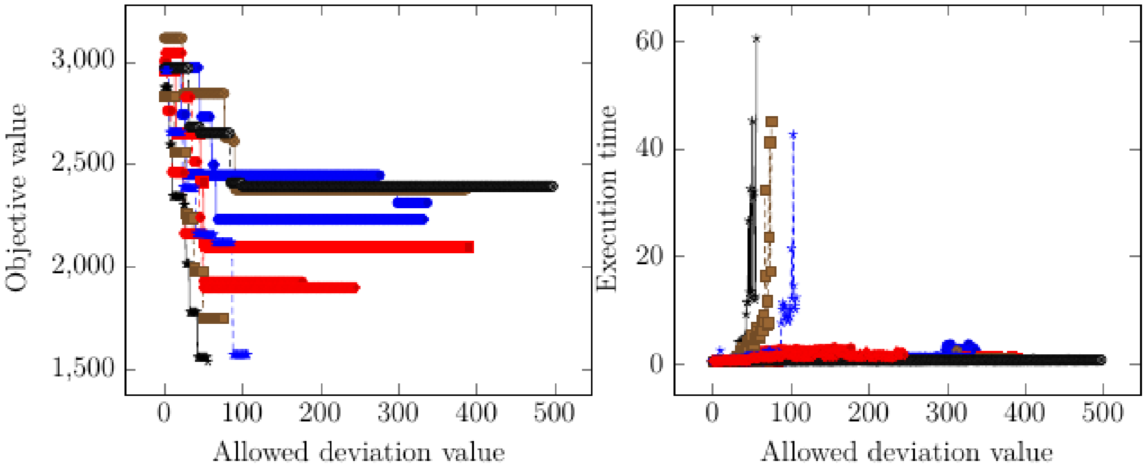
Varying the deviation's value of the travel allowed



(a) Objective

(b) Execution time

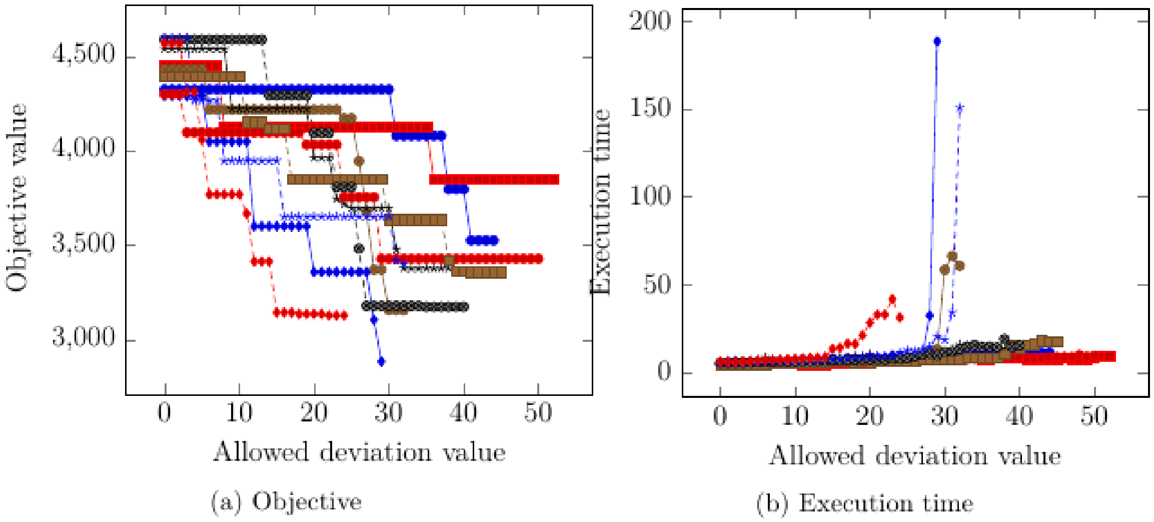
Figure 4: Vary the deviation value with all random and with 20 users



(a) Objective

(b) Execution time

Figure 5: Vary the deviation value with random close homes and with 10 users

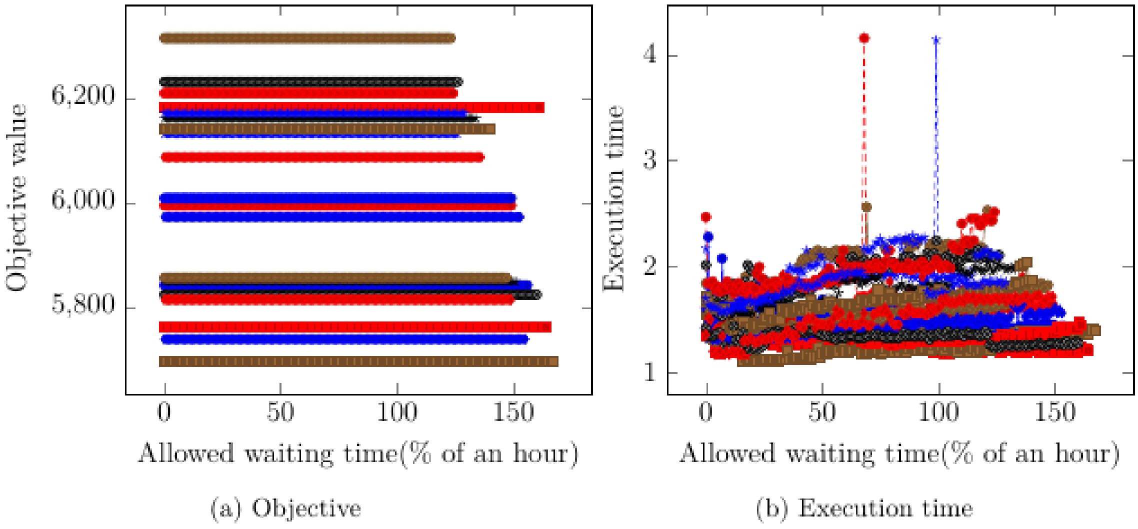


(a) Objective

(b) Execution time

Figure 6: Vary the deviation value with random close homes and with 15 users

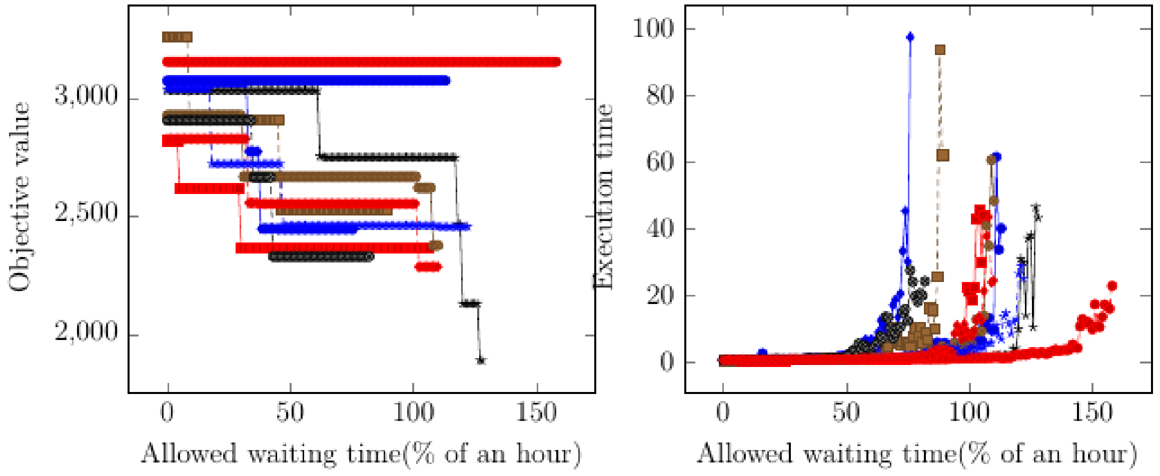
Varying the waiting time values



(a) Objective

(b) Execution time

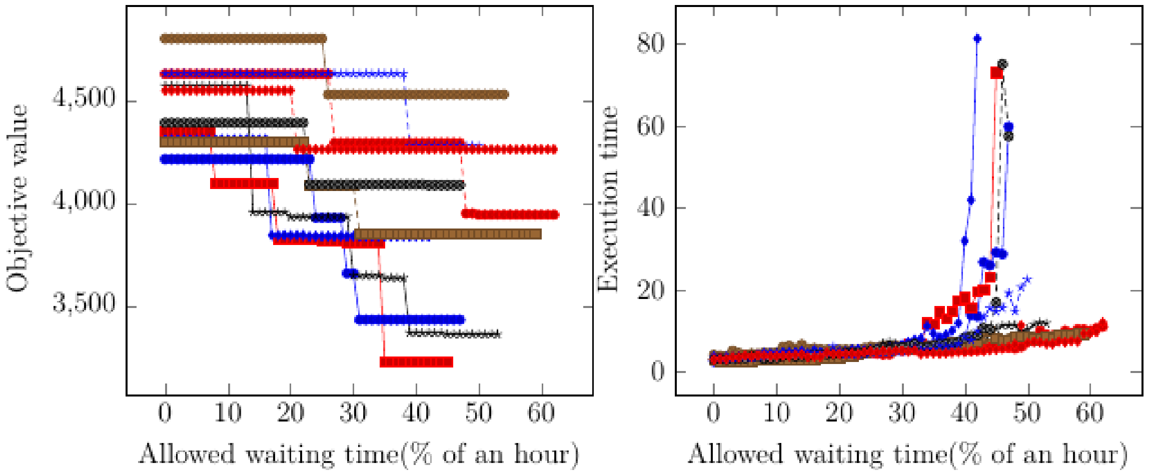
Figure 7: Vary the time windows with all random and with 20 users



(a) Objective

(b) Execution time

Figure 8: Vary the time windows with random close homes and with 10 users

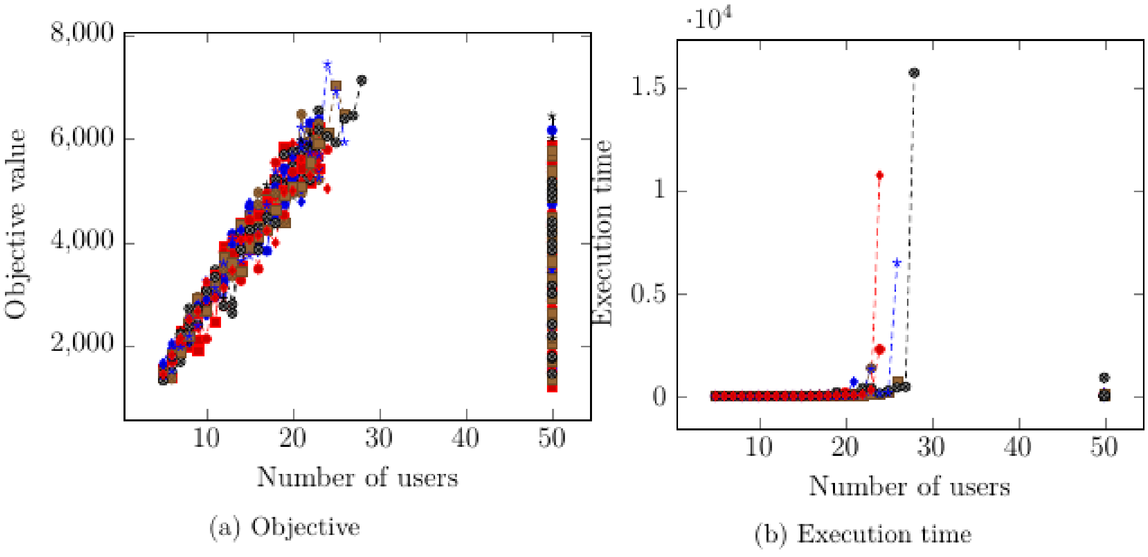


(a) Objective

(b) Execution time

Figure 9: Vary the time windows with random close homes and with 15 users

Varying the number of users



(a) Objective

(b) Execution time

Figure 10: Vary the users random close homes

Results for the city of Grenoble

Cities : VOIRON VIF CROLLES

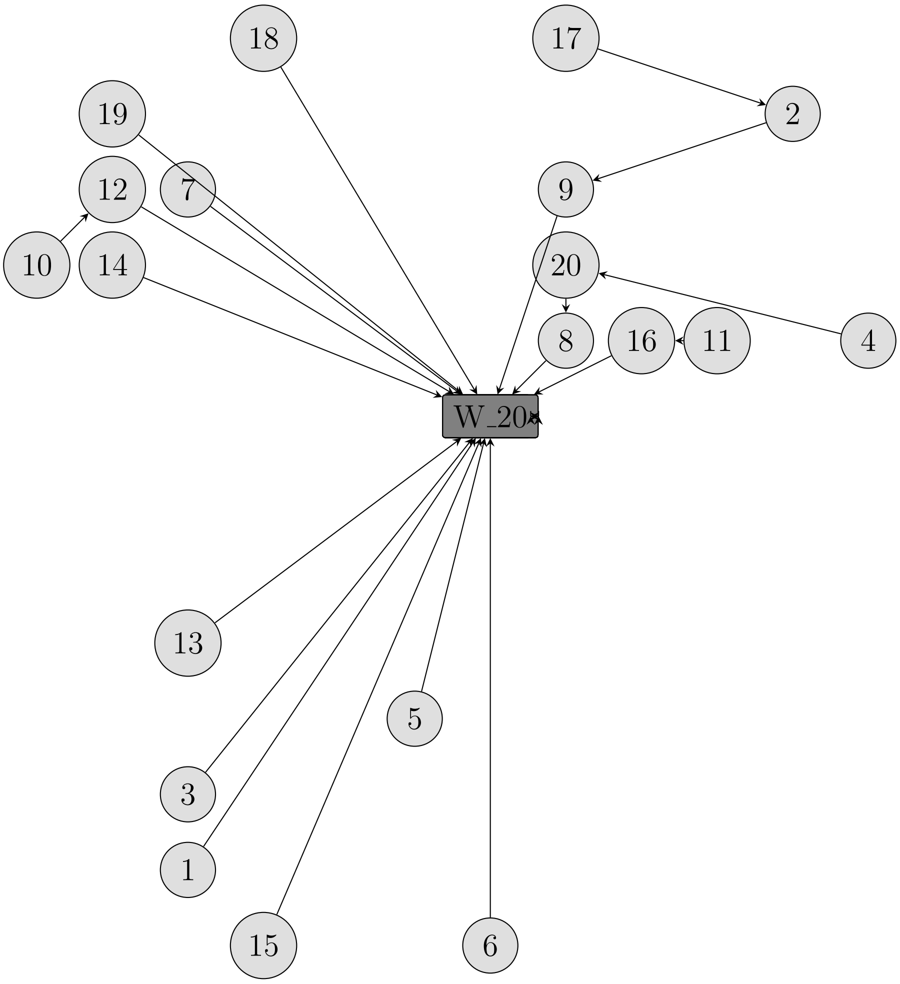


Figure 11: Home-to-work trip

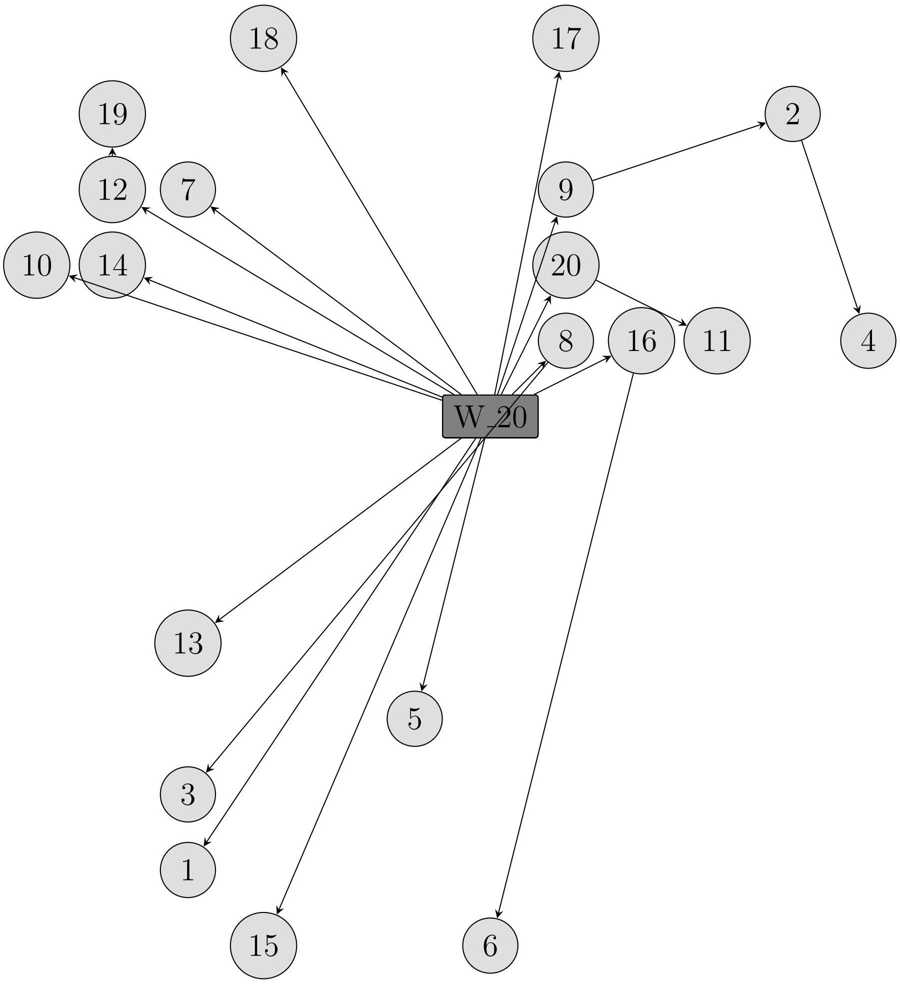


Figure 12: Work-to-home trip

Cities : VOIRON VINAY SAINT-LAURENT-DU-PONT

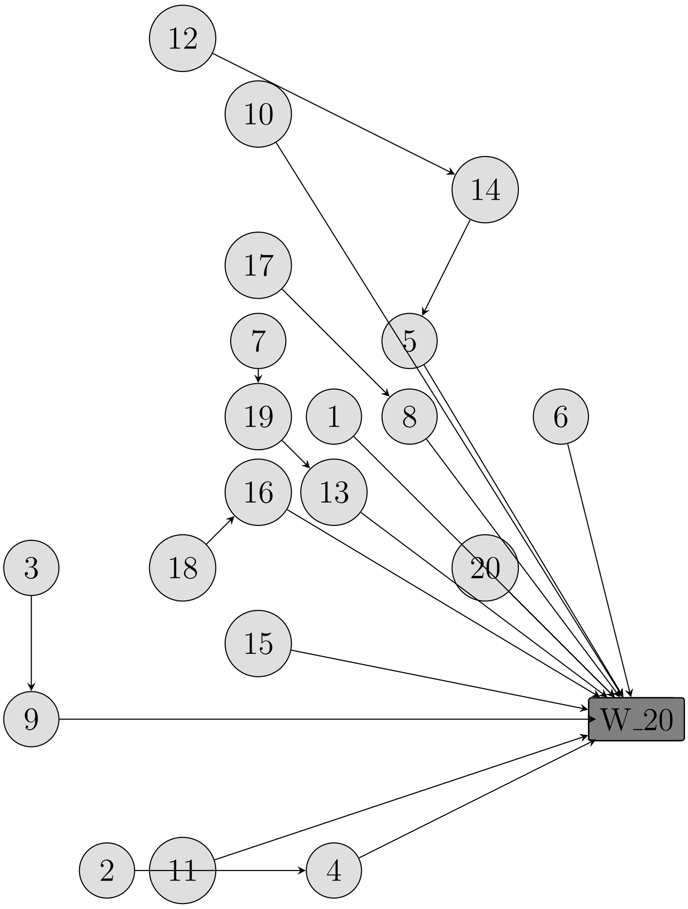


Figure 13: Home-to-work trip

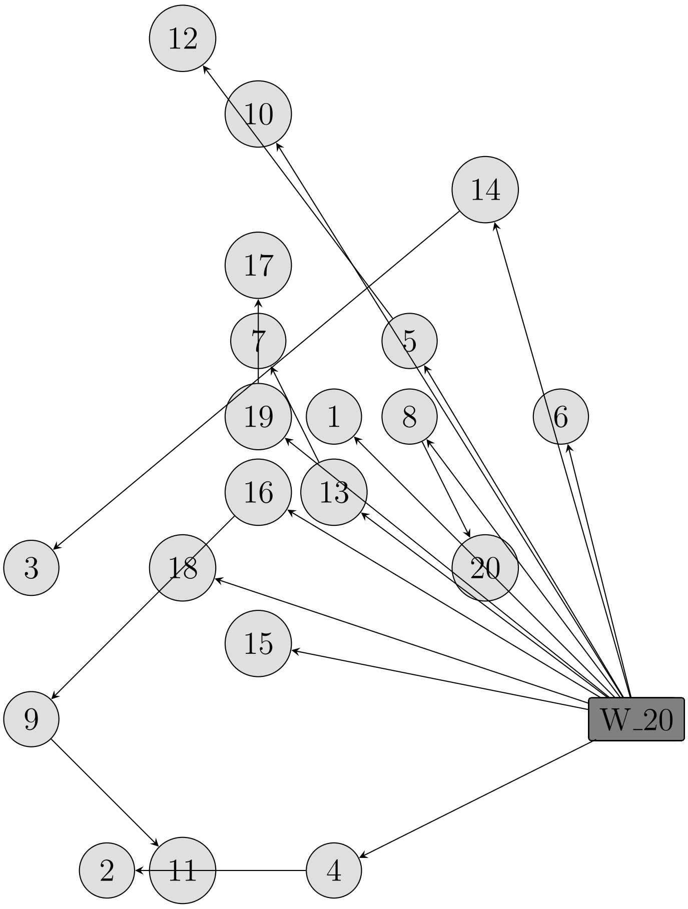


Figure 14: Work-to-home trip

Cities : PONTCHARRA LE-TOUVET CROLLES

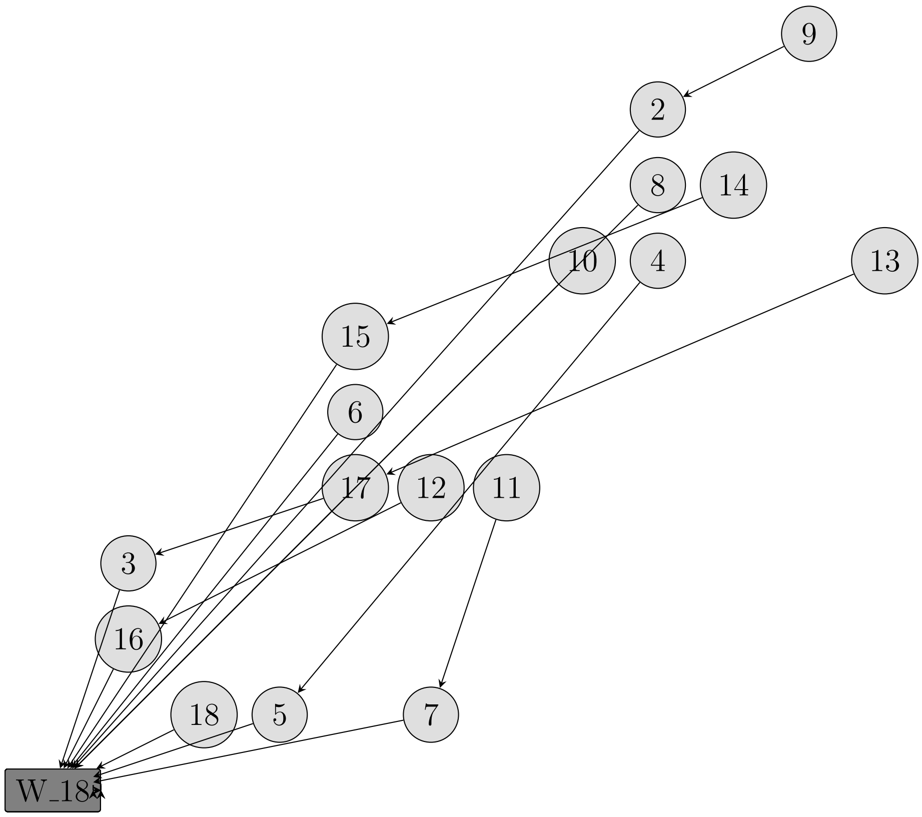


Figure 15: Home-to-work trip

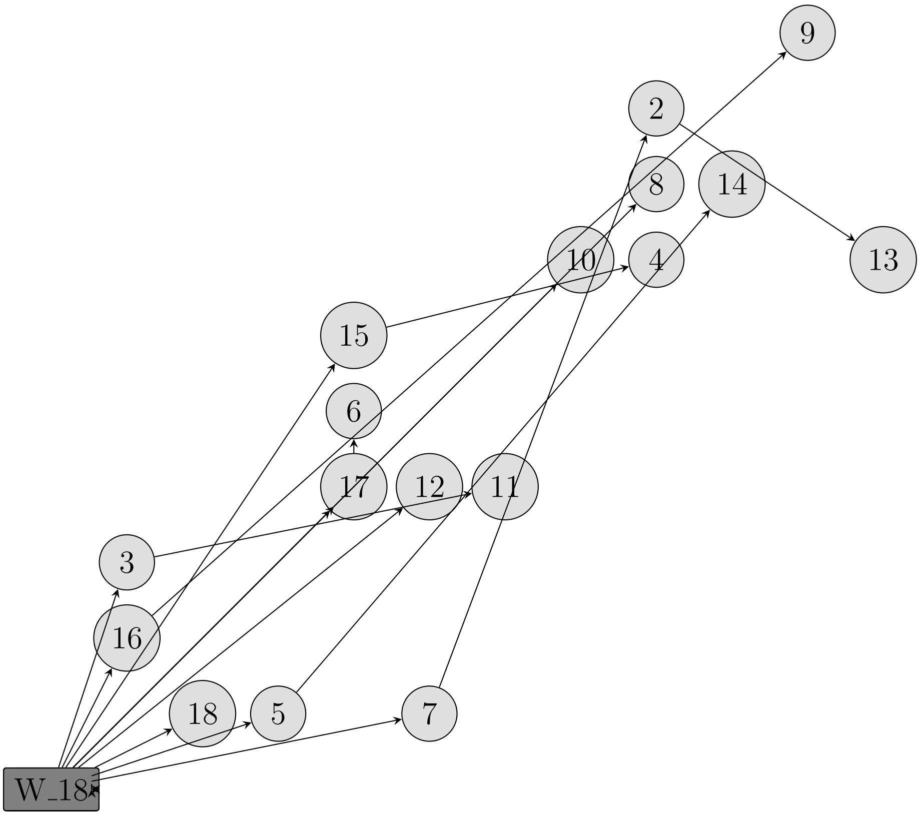


Figure 16: Work-to-home trip

Cities : VIZILLE PONT-DE-CLAIX VIF

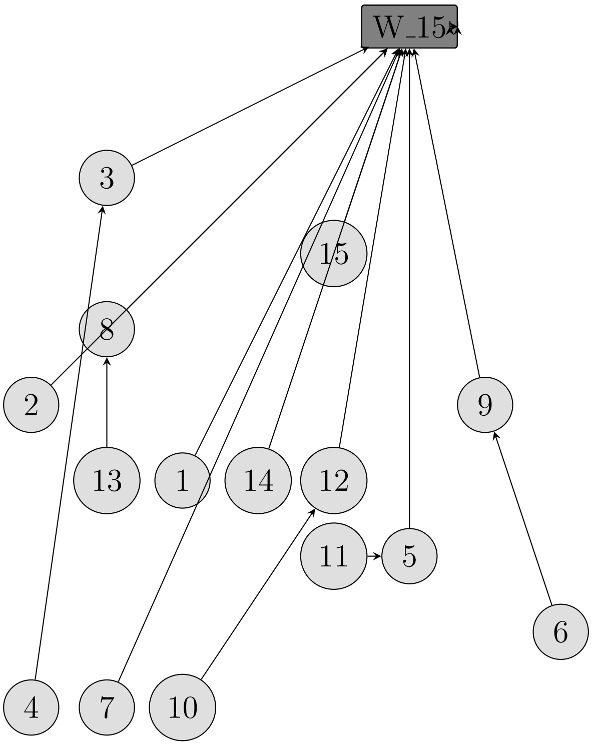


Figure 17: Home-to-work trip

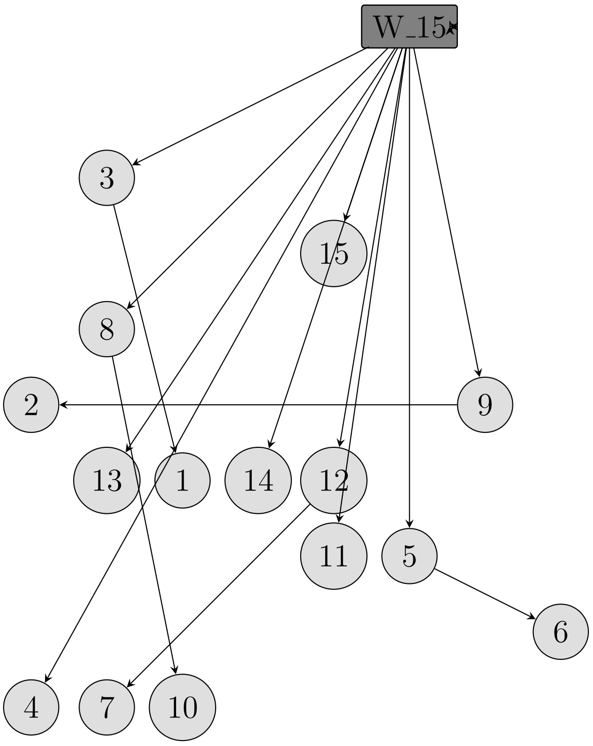


Figure 18: Work-to-home trip