Car relocation for carsharing service: Comparison of CPLEX and Greedy Search

Rabih Zakaria
OPERA, UTBM
90000 Belfort, France

Mohammad Dib
GDF SUEZ - CEEME
Paris, France

Laurent Moalic
OPERA, UTBM
90000 Belfort, France

Alexandre Caminada
OPERA, UTBM
90000 Belfort, France

***Abstract*—In this paper, we present two approaches to solve the relocation problem in one-way carsharing system. We start by formulating the problem as an Integer Linear Programming Model. Then using mobility data collected from an operational carsharing system, we built demands matrices that will be used as input data for our solver. We notice that the time needed to solve the ILP using an exact solver increases dramatically when we increase the number of employees involved in the relocation process and when the system gets bigger. To cope with this problem, we develop a greedy algorithm in order to solve the relocation problem in a faster time. Our algorithm takes one second to solve the relocation problem in worst cases; also, we evaluated the robustness of the two approaches with stochastic input data using different numbers of employees.**

I. Introduction

It is of common knowledge nowadays that dealing with environmental issues is a crucial condition for a safer thus brighter future for the generations to come. Indeed, the quality of life in cities all over the world is being seriously affected by the negative outcomes of disastrous policies that do not take into account the dangerous increase in vehicles emissions, noise pollution, congestion and lack of parking spaces. This situation has recently generated a new way of thinking among the cities towards the private use of cars by individuals, and this attitude is centered on producing better solutions for reducing car ownership while still maintaining the private cars benefits. As a concept, car-sharing service can be considered as a cornerstone of such a new way of thinking, having the ability to offer solutions to the problems mentioned earlier in an indubitably positive and advantageous way [1],[2].

Vehicle sharing can be possible through various methods that include carsharing in work institutions and neighborhoods, station cars and multiple carsharing stations [3]. The first two methods involve round trips in opposition to the third which is mostly involved in one-way trips. The said one-way trips constitute a heavier load on a service providers budget due to the need of a bigger number of personnel that will be given the task of constantly relocating vehicles [4]. The accomplishment of this task is continuously needed to ensure that there are always both parking spaces and cars available in each and every station and to avoid the risk of not being able to return a car due to over filled stations or the risk of not finding a vehicle in an empty one. Setting a balanced system is difficult since the demand varies constantly during the day. As a result, client needs will not be met all the time: empty stations will reject car requests and full ones will reject parking requests, in which case clients will have to search for other neighboring stations to give back their cars or receive ones. Solving the imbalance encountered will require the hiring of specific employees by carsharing agencies. These employees, referred to as "Jockeys" in the following, will be in charge of relocating cars to meet the needs of clients in each station meaning that they will move cars in between station to redistribute them. The ideal cars relocation will minimize, and even may get rid of, both types of rejected client requests, but achieving this goal is complicated. The cause of that complication lies in the constant fluctuation of the car rental demands and the length of the duration and distance between targeted stations. As known, limiting the number of rejected cases, to take or to return a car into stations, is crucial to define the right number of jockeys and this number may change along days and weeks.

The literature includes many works tackling this particular carsharing and bikesharing constraint [5],[6]. In [7], the author suggests the clients participation as a solution. This method has proved to be successful. It has reduced 42% of the overall number of vehicle relocation trips but it requires the contribution of every client which is not always guaranteed. [8] compared two-trip forecasting models, namely neural networks and support vector machines, in a multiple-station shared-use vehicle system, but they did not tackle the cars relocation problem. On the other hand, [9] set forth a simulation model built on the basis of two selected relocation models: inventory balancing and shortest time. This model aids multiple-station shared-use vehicle operators in recognizing the efficient resources' combination and system set up. Shortest time technique stands for moving a car from a near station in the shortest possible time. While inventory balancing technique means moving a car from overfull station to another one that needs cars. In another paper, [4] presented a decision support system for carsharing companies to determine a set of staff and operating parameters for the car relocation problem. Tested on a set of commercial network from a carsharing company in Singapore, the simulation results recommend a set of parameters which can lead to reduce the staff cost and the number of cars relocation operations.

The coming sections will firstly present the problems physical and formal aspects as well as the data used for the purpose of the study. Secondly, they will describe the optimization algorithm. After that, a comparison and an evaluation of the two approaches are presented based on the simulation results.

978-1-4799-4497-2/14/$31.00 ©2014 IEEE

II. Relocation Problem

A. Physical Description For The Relocation Problem

In our study, we consider a carsharing system where a client can take a car from a station and then, he can return it to any other one. Each station has a given capacity of cars. During the day, the number of available cars in each station will vary depending on the clients demand. Knowing the limited capacity and the number of available cars in each station at different times, we can expect that some clients demands will not be satisfied. Thus, we say that a clients demand is not satisfied or rejected in two cases:

- The client arrives at a station to take a car and cannot find any available one. We refer to this by "Demand Rejected Because a Station is Empty" or simply "RSE".
- The client arrives at a station to return the car, and cannot find an empty place. We refer to this by "Demand Rejected Because a Station is Full" or simply "RSF".

To reduce the number of rejected demands, carsharing operators recruit employees to relocate cars from full stations to other stations that need cars to satisfy client demands. The objective is to minimize the rejected demands as well as to minimize the number of relocation operations. The relocation problem can be seen as a pickup and delivery problem in a metric space where employees move cars in a time-expanded network. Each employee will follow a path where he will take a car from a station and deliver it to another one, depending on the need to reduce the number of rejected demands at different times.

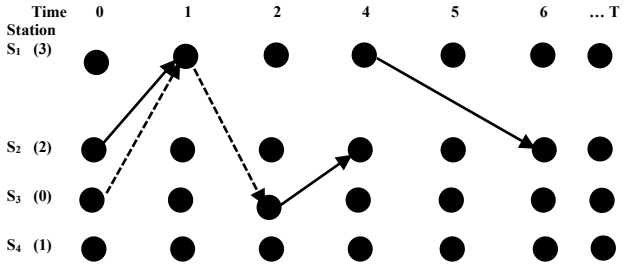

Fig. 1. Simple representation of a carsharing system in an urban area.

Figure 1 shows a representation of a simple carsharing system, which has four stations. On the first column to the left we can find the station names, beside them to the right, between parentheses, we read the number of initial available cars at each station. Each disc in the columns represents a station at time t, which is indicated in the top of the columns. The solid arrows represent the demands. The beginning of the solid arrow is a demand to take a car from the station at the time indicated in the top of that column. While the end of the solid arrow represents the demand for returning the car to the station target of the arrow at the time indicated at the top of the column. Based on the demand and the number of initial vehicles, we can calculate the number of available vehicles in each station at each time step. If the maximum number of parking spaces of a station is three, in this example, a user who will arrive at station S_1 at time t_1 will not find a place to park his car and therefore the client demand for returning a car will be rejected since the total cars in the station cannot exceed three. Another rejected demand is going to occur when a user wants to take a car from station S_3 at time t_2 since there is no available cars. To avoid these demands from being rejected, ideally a jockey should bring out a car from station S_1 at time t_1 and move it to station S_3 at time t_2.

Thus, to reduce the number of rejected demands, a jockey should relocate cars between stations. We assume that each relocation operation is performed by one jockey, and will take two time steps: the first time step to move from the starting station to the intermediate station where he will take the car, and the second time step to reach the destination station where he will return the car. For instance, we chose two time steps because in the region, subject of our study, the maximum time needed to go from a station to any other station in traffic jam situation (worst case) is one time step of 15 minutes. So for simplicity, we fixed this time for all the relocation operations.

B. Integer Linear Programming Formulation

Starting from [4], the relocation problem can be modeled as a two dimensional time-space matrix of size $N \times T$, where N is the total number of stations $S = \{1,2,..,N\}$ and T is the number of all the time steps in the day starting from 1 to T. Each element of the matrix represents a station S_i at time t. For each station $s \in S$ we created T nodes to represent that station at each time t. Then we put all the $S \times T$ nodes in one row vector $V = \{1_1, ..., i_{t-1}, i_t, i_{t+1}, N_T\}$. During the day, we consider that an employee is involved in three types of activity: Waiting, Moving and Relocation tasks. Therefore, we created three sets of arcs in the time-space network to represent these activities. For each node $i_t \in V$, we construct an arc that represents a waiting activity between i_t and i_{t+1}, we call this set $A_1 = \{..., a_1(i_t, i_{t+1}), ...\}$. Then, for each node i_t in V, we construct $N-1$ arcs to represent moving activities between station i and $j \forall i,\ j \in S,\ i \neq j$, from time step t to time step $t + t_{ij}$ where t_{ij} is the number of time steps needed to go from station i to station j, we named this set $A_2\{..., a_2(i_t, j_{t+t_{ij}}), ...\}$. In the same way of creating moving activities, we created $N-1$ arcs to represent relocation activities for each station, and we denote this set $A_3\{..., a_3(i_t, j_{t+t_{ij}}), ...\}$. We represent the available staff that will be involved in doing these activities by a set $E = \{1, ..., e, ..., W\}$ where W is the maximum number of available employees. We have formulated our relocation problem as an Integer Linear Programming Model. We used six types of decision variables:

- u^e : Binary variable, that takes the value of one if the employee e has been used during the day and zero otherwise.
- $wait^e_{i_t i_{t+1}}$: Binary variable associated with the set of waiting activities A_1. It takes the value of one if employee e has been waiting at station i from time step t to $t+1$ and zero otherwise.
- $move^e_{i_t j_{t+t_{ij}}}$: Binary variable associated with the set of moving activities A_2. It takes the value of one if employee e has been moving from station i to station j, from time step t to $t + t_{ij}$ and zero otherwise.

- $rel^{e}_{i_t j_{t+t_{ij}}}$: Binary variable associated with the set of relocation activities A_3. It takes the value of one if employee e has been relocating a car from station i to station j, from time step t to $t+t_{ij}$ and zero otherwise.
- $out^{r}_{i_t}$: Integer variable to represent the number of rejected demand to take a car out of a station i at time step t.
- $in^{r}_{i_t}$: Integer variable to represent that number of rejected demand to return a car into a station i at time step t.

In the other hand, here are our constants that we will be used as input for our model:

- av_0 : Represents the number of Available Vehicles at time step 0 in each station i.
- out_{i_t} : Represents the number of demands to take a car out of a station i at time step t.
- in_{i_t} : Represents the number of demands to return a car into a station i at time step t.
- p_i : Number of parking slots in each station i.
- c_{ij} : Cost of a moving or relocating activity from station i to station j.
- c_e : Cost of using one staff during the day.
- c_{in} : Cost of rejecting a client demand for returning a car into a station.
- c_{out} : Cost of rejecting a client demand for taking a car from a station.

In addition, we used one dependent variable:

- av_{it} : Number of available vehicles at station i at time step t.

The ILP model for the problem is:

$$\begin{aligned} Min \;\; Z = c_{ij}\Big(& \sum_{(i_t, j_{t+t_{ij}})\in A_3} \sum_{e\in E} move^{e}_{i_t j_{t+t_{ij}}} \\ & + \sum_{(i_t, j_{t+t_{ij}})\in A_4} \sum_{e\in E} rel^{e}_{i_t j_{t+t_{ij}}}\Big) + c_{out} \sum_{i_t \in V} out^{r}_{i_t} \\ & + c_{in} \sum_{i_t \in V} in^{r}_{i_t} + c_e \sum_{e\in E} u^{e} \end{aligned} \tag{1}$$

Subject to:

$$\begin{aligned} & \sum_{i\in S} wait^{e}_{i_1 i_2} + \sum_{\substack{i,j\in S \\ i\neq j}} move^{e}_{i_1 j_{1+t_{ij}}} + \sum_{\substack{i,j\in S \\ i\neq j}} rel^{e}_{i_1 j_{1+t_{ij}}} \\ & = u^{e} \qquad\qquad \forall e \in E \end{aligned} \tag{2}$$

$$\begin{aligned} & wait^{e}_{i_{t-1} i_t} + \sum_{(j_{t-t_{ij}}, i_t)\in A_3} move^{e}_{j_{t-t_{ij}} i_t} \\ & + \sum_{(j_{t-t_{ij}}, i_t)\in A4} rel^{e}_{j_{t-t_{ij}} i_t} - wait^{e}_{i_t + i_{t+1}} \\ & + \sum_{(i_t, j_{t+t_{ij}})\in A3} move^{e}_{i_t j_{t+t_{ij}}} + \sum_{(i_t, j_{t+t_{ij}})\in A4} rel^{e}_{i_t j_{t+t_{ij}}} \\ & = 0 \qquad\qquad \forall\, i_t \in V,\, e \in E, t > 1 \end{aligned} \tag{3}$$

$$\begin{aligned} & av_{i_t} = av_{i_{t-1}} + (in_{i_t} - in^{r}_{i_t}) - (out_{i_t} - out^{r}_{i_t}) \\ & + \sum_{(j_{t-t_{ij}}, i_t)\in A_4} \sum_{e\in E} rel^{e}_{j_{t-t_{ij}} i_t} - \\ & \sum_{(i_t, j_{t+t_{ij}})\in A_4} \sum_{e\in E} rel^{e}_{i_t j_{t+t_{ij}}} \qquad \forall\, i_t \in V \end{aligned} \tag{4}$$

$$av_{i_t} \leq p_i \;\; \forall\, i_t \in V \tag{5}$$

$$in^{r}_{i_t} \;\leq\; in_{i_t} \;\forall\, i_t \in V \tag{6}$$

$$out^{r}_{i_t} \;\leq\; out_{i_t} \;\forall\, i_t \in V \tag{7}$$

$$u^{e} = (0,1) \;\; \forall \;\; e \in E \tag{8}$$

$$wait^{e}_{i_t i_{t+1}} = (0,1) \;\; \forall\, (i_t, i_{t+1}) \in A_1,\; e \in E \tag{9}$$

$$move^{e}_{i_t j_{t+t_{ij}}} = (0,1) \;\; \forall\, (i_t, j_{t+t_{ij}}) \in A_2,\; e \in E \tag{10}$$

$$rel^{e}_{i_t j_{t+t_{ij}}} = (0,1) \;\; \forall\, (i_t, j_{t+t_{ij}}) \in A_3,\; e \in E \tag{11}$$

$$in^{r}_{i_t} \geq 0 \;\; \forall\, i_t \in V \tag{12}$$

$$out^{r}_{i_t} \geq 0 \;\; \forall\, i_t \in V \tag{13}$$

$$av_{i_t} \geq 0 \;\; \forall\, i_t \in V \tag{14}$$

The objective function (1) minimize the number of rejected demands and the number of relocation operations needed to reduce the number of rejected demands. Constraint (2) is used to make sure that each employee is involved in just one activity at a time and it is used to set the variable u^e to one if the employee e is used at $t = 1$. Constraint (3) is used to make sure that an employee will not start a new activity until he finished the previous one. We used Constraint (4) to calculate the number of available vehicles at each station at each time step. It depends of the number of available vehicles in the previous time step, the number of vehicles moving in/out of station and the number of vehicles relocated in/out of the station. Constraint (5) is used to make sure that the number of available vehicles at a station cannot be greater than the capacity of the station. Constraints (6) and (7) ensures that the number of rejected demands at a station cannot exceed the demand itself. Constraints (8)-(11) forces their variables to take binary values, while constraints (12)-(14) make sure that their variables are non-negative.

III. Mobility Data

In this section, we will describe the mobility data that we use as input for our relocation model. In order to generate our mobility data, we used the platform GeoLogic developed by our team in [10]. This platform takes advantage of surveys data and socio-economical information collected by professionals in regional planning, in addition to GIS shape files describing the geographic zone of our study. Collected data belongs to a geographic zone in Paris of $20Km \times 10Km$, which is represented by a grid of equal cell size. Using all this information and based on the preferred parameters of the carsharing organization, we could construct our mobility data. Flows of mobility data are represented by a 3d matrix $F = (f_{i,j,t})$ where $f_{i,j,t}$ represents the number of people moving from cell i to cell j at time period t. The platform uses the mobility data to locate the stations for the carsharing system in the zone of our study.

The backbone of the platform consists of a multi objective local search algorithm that tries to locate the stations in a way that covers the maximum user demands in an appropriate manner. The algorithm tries to optimize three objectives:

1) f_1: flow maximization i.e. the locations must allow us to maximize the flows between themselves.
2) f_2: balance maximization i.e. the location must allow us to maximize the balance between inflows and outflows of a station.
3) f_3: minimization of flow standard deviation i.e. the location must allow us to get a uniform flow along the day.

The algorithm for locating the stations and the evaluation of user's demand during each time step, supposes a good understanding of who will use the service. This part was done with the operator who will deploy the carsharing service. Among all the mobility flows, some filters are used for selecting the rate of users who will use the service depending on their profile (age, sex, etc.). Moreover, we defined a capture radius for each station, which defines the maximum distance within it, users are ready to walk to reach the station. Based on this radius value, a station st_i can cover one or more cells. In territory planning it is generally considered that one can walk 300m for taking a public transportation.

Using the mobility data, we located 20 stations in the region where we did our study; we assumed that each station contains 10 parking spaces. Hence, each station can contain 10 cars at maximum. Then using the demand data, we calculate the expected rejected demand in each station that we used as input data for our model.

Furthermore, to decide the client assignment to the cars, we used a probabilistic distribution on the demand data. This distribution is fixed by sociological survey and input hypothesis done by the carsharing exploitation company. After that, using the client assignment to the cars and the demand data from each station to all other stations at each time step we were able to construct two new matrices:

out: Contains the number of cars that will be taken from each station at each time step.

in: Contains the number of cars that will be dropped off into each station at each time step.

Knowing the number of vehicles that will be taken and returned in each station at each time step, we could calculate the number of available vehicles in each station at each time step. However, since each station has a limited capacity, and sometimes there are no available cars, some demands will be rejected. To calculate the number of rejected demands in each station at each time step without using any employee for relocating cars, we used some constraints from the ILP described earlier in this paper. When we set the number of employees to zero, constraint (4) will be simplified to take the form below:

$$av_{i_t} = av_{i_t-1} + (in_{i_t} - in^r_{i_t}) - (out_{i_t} - out^r_{i_t}) \qquad (15)$$

Using this new constraint, we were able to construct two new matrices for rejected demands:

$out^r_{i_t}$: Matrix of rejected demands to take a car from a station i at time step t. This type of rejected demand occurs because the station i is empty at time step t, so the demand will not be satisfied.

$in^r_{i_t}$: Matrix of rejected demands for returning a car into a station i at time step t. This type of rejected demand occurs because the station i is full at time step t, so the demands to return cars, will not be satisfied.

These new matrices will reflect the number of rejected demands in each station during each 15 minutes of the day.

A. Input Data For The Model

We used the platform to locate the stations and to generate the demand data for our carsharing system. We will use the demand data as input for our relocation model. For this sake, we work on four parameters in the platform:

1) Number of stations
2) Number of trips per car
3) Number of parking spaces in each station
4) Number of cars in the system

For our first data set, we started by generating the demand data for a carsharing system, which is composed of 20 stations with 10 parking spaces for each station and 150 cars with an average of 6 trips per car. These values are chosen by the carsharing organization. We ran our algorithm using the generated data set as input data, and then we compared the result of the algorithm with the results of the exact solver for our ILP. After that, we used our platform to generate several configurations for our carsharing system. When we increase the number of stations, we obtain a bigger carsharing system that covers more user demands. The variation in the number of parking spaces leads to different station size which results to less rejection of user demands to return a car into stations. In addition, we varied the average number of trips per car to evaluate the effect of this factor on the relocation operations, while the number of cars in each station is fixed as chosen by the carsharing company.

IV. A Greedy Algorithm For The Relocation Problem

In order to solve the relocation problem in a faster way, we developed a greedy algorithm to reduce the number of rejected demands. The algorithm tries to solve the maximum number of rejected demands using the minimum number of relocation operations. At each time step, the algorithm tries to find the best path that solve the maximum number of rejected demand and assign it to the jockey. For example, if we expect a Rejected demand because a Station is Empty or simply a "RSE" at time $t+2$, the jockey should bring a car to the empty station to fulfill the upcoming request. That is, the jockey should look for a station that has available car at $t+1$ to drive it to the required destination station at $t+2$. On the other side, if we expect a Rejected demand because a Station is Full or simply "RSF" at $t+1$, a jockey should drive a car from the full station to another one that would need a car or has an empty space at $t+2$.

The relocation operation must be optimized in order to solve as much rejected demand as possible. The best the jockey

can do is to solve two rejected demands in one relocation operation. For instance, this situation occurs when we have a "RSE" at $t+2$ and a "RSF" at $t+1$. In this case, the jockey goes to the station that has a "RSF", and drives a car to the station that has a "RSE", in this way the jockey solves two rejected demands by one relocation operation. However, this situation does not often occur. In other situation, we only have one rejected demand between $t+1$ and $t+2$, so in this case, the jockey will solve this rejected demand and in the same time will try to anticipate and solve another upcoming rejected demand (which may appear at time greater than $t+2$). Moreover, we have another situation where we do not have any rejected demand between $t+1$ and $t+2$. In this case, the jockey tries to move cars between the stations at these times in order to anticipate and solve rejected demands before their occurrence in the future. Thus, to make the algorithm more general and then more efficient, at each time the algorithm will look for the soonest upcoming "RSF" and "RSE" and try to solve them as it is indicated in Figure 2.

Each relocation operation consists of a path from the origin station where the jockey starts the operation to the intermediate station where he will pick up a car and drive it to the destination station. After proceeding with the relocation operation, we update the number of cars in affected stations. We keep in mind that a relocation operation can raise rejected demands in the related stations in the future time, so when taking the decision to relocate a car we must be cautious as to not generate much more rejected demands. During each relocation operation, the jockey follow a path of three stations:

1) Starting station where the jockey starts the relocation operation at relocation starting time.
2) Intermediate station where the jockey arrives to pick up a car at relocation intermediate time.
3) Destination station where the jockey arrives to deliver the picked up car at the relocation destination time.

When a jockey is willing to start a relocation operation, he must choose a path between different possibilities. He starts from one origin station. However, the jockey may have different possibilities to choose an intermediate and a destination station. A jockey should consider the path of minimum cost, which is the one that solve the maximum of rejected demands in the soonest delays. Since we considered that we need one time step to move from any station to any other station, we did not integrate the distance in our calculation. However, we can add it to our formula by simply adding the distance factor to our cost function.

We use a cost function to choose the paths that solve the maximum rejected demands in a way that we privilege the paths that solve the soonest upcoming rejected demands. To calculate the minimum path cost we use this formula:

$$cost = \frac{1}{E-G+1} + \Delta \tag{16}$$

Where:

$$\Delta = \frac{1}{t_G+1} \times G - \frac{1}{t_E+1} \times E \tag{17}$$

Δ : is used to increase the cost of paths having generated demands, and to privilege the path that generates a late rejected demand on the path that generates sooner rejected demands.

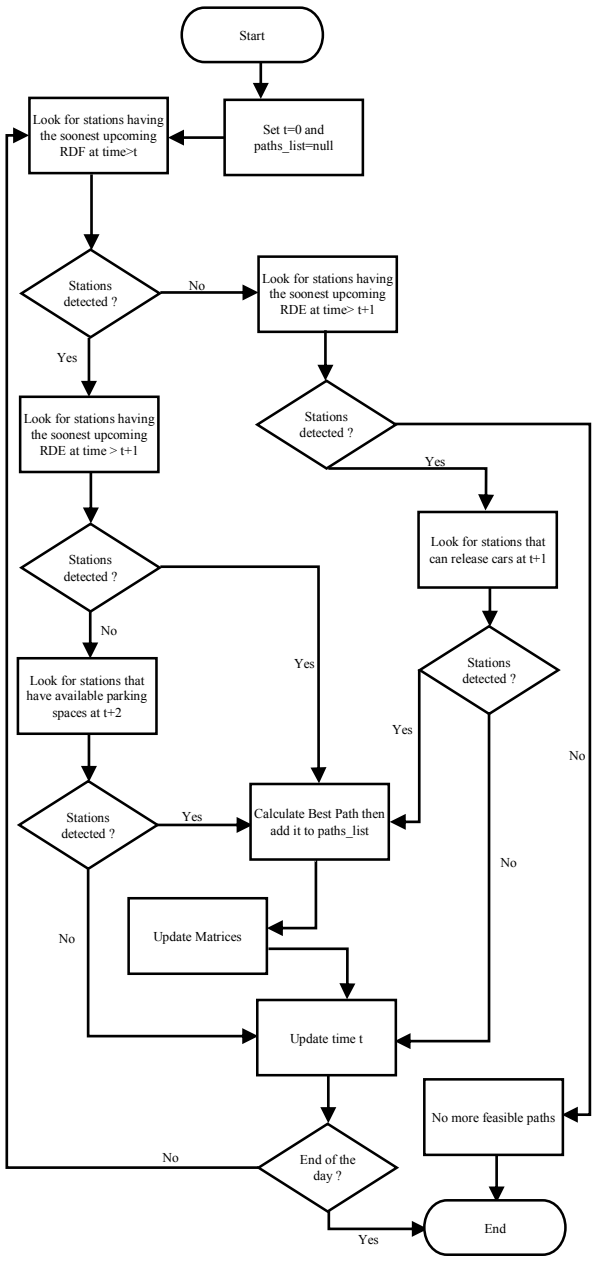

Fig. 2. Flowchart of the Relocation Algorithm

E : Number of Eliminated Rejected Demands, $E \in [1;2]$.

G : Number of Generated Rejected Demands, $G \in [0;1]$.

t_G : Index of time steps when the demands might be generated, $t_G \in]1;96[$.

t_E : Index of time steps when the demands might be eliminated, $t_E \in]1;96[$.

When the number of eliminated rejected demands E increase, the cost value will increase and vice-versa. If we have to choose between two paths that solve the same number of rejected demands but in different time steps, we will select the path that will eliminate the soonest upcoming rejected demands. In other hand, if we have to choose between many

paths that eliminate one rejected demand and generate another, we will choose the path that will solve the soonest upcoming rejected demand and generate the latest upcoming rejected demand.

At $t = 0$, the algorithm starts by looking for the stations that have the soonest upcoming "RSF" at time greater than the current time, to give the jockey the required time to arrive at the station. If it finds them, then as second step, it looks for the stations that have the soonest upcoming "RSE" at time greater than the current time plus one time step, to give the employee the necessary time to go from the intermediate station to the destination station. If he finds those, then the algorithm calculates the minimum path cost, after that it adds it to the paths list of the involved jockey.

If in the second step, the algorithm did not find any "RSE", then it will look for all stations that have empty space to receive a car at $t+2$. When the algorithm gets the list of these stations, it will choose the minimum path cost and add it to paths list.

If in the first step, the algorithm did not find any station that has a "RSF" at $t+1$, then it looks for all the stations that have "RSE" at $t+2$. If it finds them, then, as second step, the algorithm looks for the stations that can release cars at $t+1$. If it finds those, then the algorithm calculates the minimum path cost, and add it to the paths list. If the algorithm did not find any rejected demand or it reaches the end of the day, the algorithm will stop.

By this way, the algorithm will try to anticipate any rejected demand that will occur later by moving the cars between the stations that will have rejected demands in the nearest future, in order to prevent these demands from being rejected. Even though a relocation operation must solve rejected demands, it can also causes some other demands to be rejected. For example, if we relocate a car from station st_0 at $t+1$ that can release cars to another station st_5 at $t+2$ that has "RSE", in this case we solve one rejected demand in st_5 at $t+2$. However, we may cause a new rejected demand in st_0 in the future, if the number of available cars at st_0 is zero at time ¿ $t+1$. In addition, this could happen if we relocate a car from station st_1 at $t+1$ that has a "RSF" to a station that has free parking spaces.

To use several jockeys, we run the algorithm iteratively for each jockey using the previous output as input to the next iteration; the final number of jockeys to use, is decided by the exploitation company.

V. Results And Experimentation

In this section we present a comparison between the performance of the two approaches we used in our study in reducing the number of rejected demands. In the other side, we compare the execution time of each method.

We solve our ILP model using IBM CPLEX solver on an Intel core i5 machine of 16 GB RAM. In our study, we solved each of the 27 generated configurations of our carsharing system using different number of jockeys. For each configuration we started by solving the relocation problem by using one jockey then we progressively increased the number of jockeys until all the rejected demands are solved. At each time, we were recording the execution time that takes CPLEX

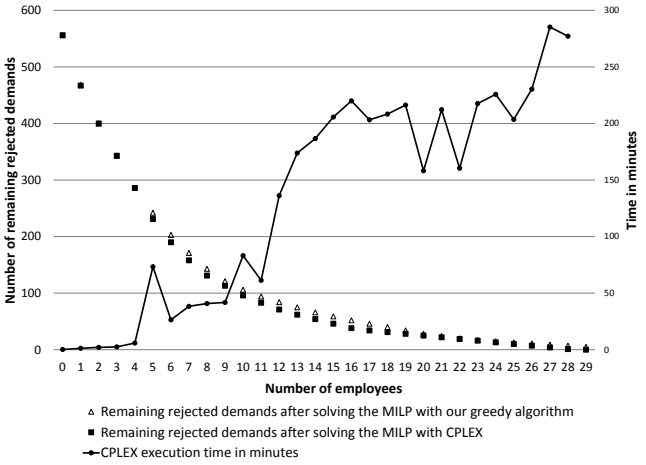

Fig. 3. ILP VS our Greedy Algorithm results(20 stations, 10 places per station and 9 trips per car)

and the greedy algorithm in order to solve our ILP model. We noticed that the execution time using CPLEX varies for each configuration size and for each number of jockey involved in the relocation operation. CPLEX takes less than one minute to solve some configurations, while in other configurations we could not get a solution using the same machine even after 27 hours. In the other side, we can observe that in all the configurations that we tried to solve, generally as much as we increase the number of jockeys the execution time tends to increase exponentially as we can see in Figure 7. Moreover, we noticed that when we try to solve the configurations that have a bigger number of stations (100), the execution time increases much faster than the other configurations that has a smaller number of stations, and we often could not get a solution for these data sets after a long time of execution using CPLEX. However, the greedy algorithm solves the most complex carsharing configuration in less than one second.

A. Greedy Algorithm VS Exact Solver Results

To evaluate our two approaches, we started by a data set which represents a carsharing configuration that has 20 stations with 10 parking places for each station and an average of 9 trips per car during the whole day. Before executing the algorithm, we had 59/96 time steps having rejected demands in this data set. Therefore, if we are using just one jockey, in the best cases, he can solve 96 rejected demands, knowing that the best he can do is solving two rejected demands in one relocation operation by moving a car from a station that has a "RSF" to a station that has a "RSE". However, the number of solved rejected demands will decrease when the number of time steps having rejected demands decreases. We consider that the time needed to move from a station to another one is constant(one time-step), for simplicity reasons we did not take into account the distance inter station and the traffic jam problems.

The graph in Figure 3 shows the total number of remaining rejected demands compared to the number of jockeys used by applying our two approaches. Our greedy algorithm shows very good results when we compare it to the optimal solutions

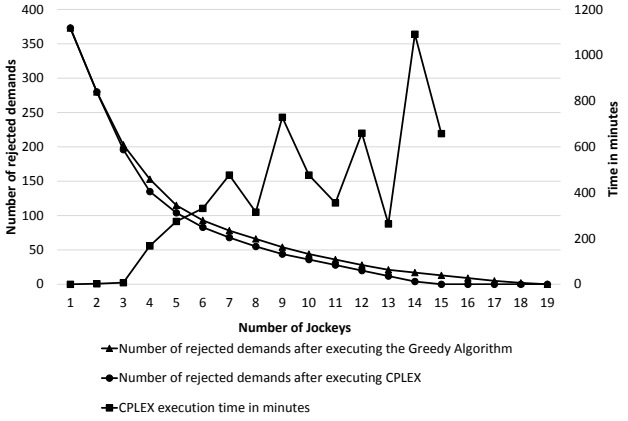

Fig. 4. ILP VS our Greedy Algorithm results(50 stations, 10 places per station and 6 trips per car)

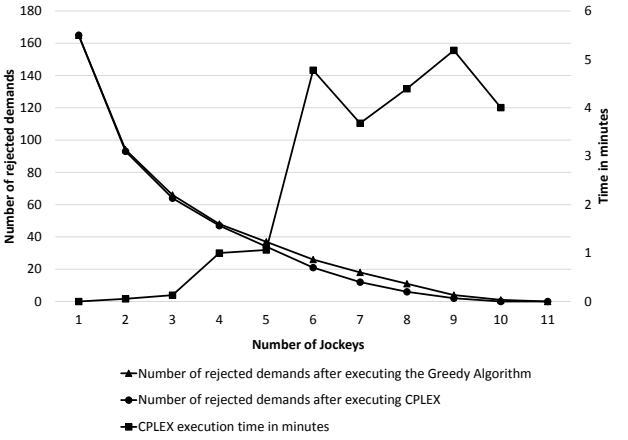

Fig. 5. ILP VS our Greedy Algorithm results (20 stations, 6 places per station and 15 trips per car)

solved by the ILP model. After execution of our greedy algorithm, the results show that when we use 10 jockeys, we could solve 81% of the rejected demands, while our ILP model solved nearly 82.5%. These results reveal the good performance of our greedy algorithm. Figure 4 and Figure 5 are other examples that prove the good performance of our greedy algorithm when we compare the exact solver method. In these two examples we can also see that there is an ascending trend in the execution time when using CPLEX while the execution time of our algorithm is often less than one second.

B. Stochastic Data

In our study, we considered that we know all the demand of cars that will be taken off and returned into the stations since the first time step of the day until the last one. However, in real life, it is impossible to know exactly what will happen during the day and thus it's very hard to predict the state of the system at each moment. For this sake, we added some stochastic variation to the original data by applying a gaussian noise on the demand.

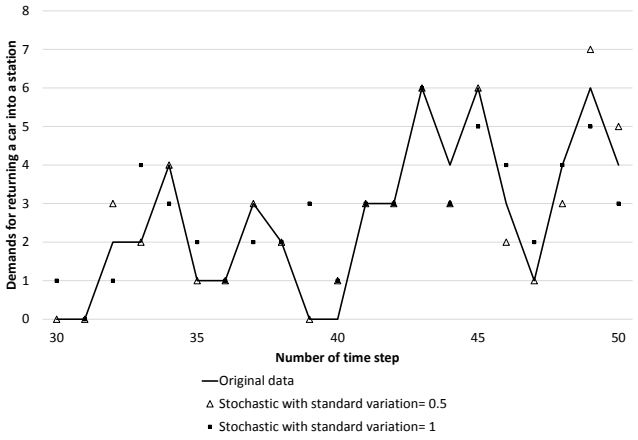

Fig. 6. Comparison between Original and stochastic demands for returning a car into a station from time step 30 to 50

In Figure 6, we see three curves of the demands to return a car into a station. The curve of original data represents the data that we used without any stochastic modifications. The other two curves represent the same demand at the station, but after applying small modifications to the values of demand. We used the Gaussian to add some slight modifications to the original data with the intention of measuring the effect of the stochastic data on the robustness of the two approaches used in this study. We chose two small standard deviation values (0.5 and 1) as we can see in Figure 6. It's important to note that the purpose of adding the Gaussian noise is just to evaluate the robustness of the obtained solutions against minor changes in the original data. After executing our greedy algorithm and the ILP model on the original data, we obtained the routes that represent the relocation operations and moves to be accomplished by the jockeys to reduce the number of rejected demands. To evaluate the robustness of the relocation algorithm and the ILP model on the stochastic modified data, we applied the obtained solutions of the original data, directly on the modified problem in order to calculate the new values of rejected demands. Results in Figure 7 show that even a slight modification in the source data could lead a bad performance in solving the relocation operations when we follow the same optimized routes generated upon solving the relocation problem on the original data using either the greedy algorithm or the ILP model. The difference between the number of rejected demands remaining after solving the relocation operation using the original data and using the modified data started to be small and then the gap gets bigger as the number of employee is increased. This result can be explained by the fact that the modifications in the data are propagated after the intervention of each employee. We can see that the number of remaining rejected demands when the standard deviation is one is more than the double of the number of remaining rejected demands when the standard deviation is a half. Hence, the relocation operations should be dynamic (real time) to be able to solve the maximum number of rejected demands when some stochastic variation occurs on the demand.

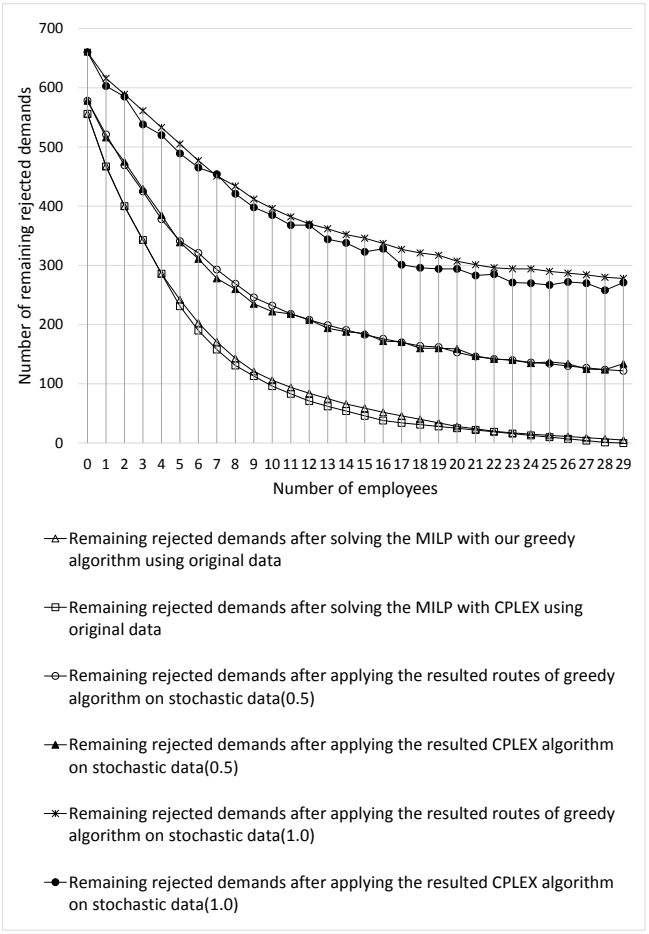

Fig. 7. Effects of stochastic data on the robustness of relocation operations

VI. Conclusion And Future Works

As mentioned previously, in this study, we started by presenting the relocation problem for a carsharing system. Then we formulated the problem as a Integer Linear Programming Model. After that, we described the mobility data and our greedy algorithm that we used to relocate the cars between the stations in a carsharing system in order to maintain the balance and then maximize the client demand satisfaction. The algorithm tries in each relocation operation to solve the maximum number of rejected demands in a minimum path cost either by resolving the rejected demands directly or by making anticipated car movements that solve rejected demands before their occurrence.

The greedy algorithm showed good performance in comparison to the ILP model. It finds competitive solutions in a less than one second while the exact ILP solver may take more than one day to solve the big data set. We also observed an ascendant trend in the CPLEX execution time when we increase the number of jockeys as well as when we increase the system size. In addition, it was clear that the effort needed for the cars' relocation, varies during the day, so we do not need the same number of jockey during inactivity periods, this will help in improving the service and in reducing the relocation cost for the carsharing operators. In other hand, we compared the robustness of the two approaches of solving the relocation problem while using data that has been slightly modified by a Gaussian noise. The results show that the efficiency of both approaches drops dramatically even when the stochastic modification is minor. These results proof that the offline routes planning for the relocation problem will not be very efficient in reducing the rejected demands in real life when stochastic data prevails. However, the rapidity of the algorithmic approach seems promising in solving the relocation problem using an online dynamic approach.

In future work, we will model a simulation for the relocating operations, which takes into account the distance and the time to move from station to another station, and simulate the impact of each relocation operation, on the whole system.

References

[1] A. M. Ball, *Car-Sharing: Where and how it succeeds*. Transportation Research Board, 2005, vol. 108.

[2] G. H. d. A. Correia and A. P. Antunes, "Optimization approach to depot location and trip selection in one-way carsharing systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 233–247, 2012.

[3] M. Barth and S. A. Shaheen, "Shared-use vehicle systems: Framework for classifying carsharing, station cars, and combined approaches," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1791, no. 1, pp. 105–112, 2002.

[4] A. G. Kek, R. L. Cheu, Q. Meng, and C. H. Fung, "A decision support system for vehicle relocation operations in carsharing systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 1, pp. 149–158, 2009.

[5] C. Contardo, C. Morency, and L.-M. Rousseau, *Balancing a dynamic public bike-sharing system*. CIRRELT, 2012.

[6] M. Benchimol, P. Benchimol, B. Chappert, A. De La Taille, F. Laroche, F. Meunier, L. Robinet *et al.*, "Balancing the stations of a self-service bike hire system," *RAIRO-Operations Research*, vol. 45, no. 1, pp. 37–61, 2011.

[7] M. Barth, M. Todd, and L. Xue, "User-based vehicle relocation techniques for multiple-station shared-use vehicle systems," *Transportation Research Record*, vol. 1887, pp. 137–144, 2004.

[8] R. L. Cheu, J. Xu, A. G. Kek, W. P. Lim, and W. L. Chen, "Forecasting shared-use vehicle trips with neural networks and support vector machines," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1968, no. 1, pp. 40–46, 2006.

[9] A. G. Kek, R. L. Cheu, and M. L. Chor, "Relocation simulation model for multiple-station shared-use vehicle systems," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1986, no. 1, pp. 81–88, 2006.

[10] L. Moalic, A. Caminada, and S. Lamrous, "A fast local search approach for multiobjective problems," in *Learning and Intelligent Optimization*. Springer, 2013, pp. 294–298.