GREEDY ALGORITHM FOR RELOCATION PROBLEM IN ONE-WAY CARSHARING SYSTEMS

Rabih Zakaria, Laurent Moalic, Alexandre Caminada, Mohammad Dib

GREEDY ALGORITHM FOR RELOCATION PROBLEM IN ONE-WAY CARSHARING SYSTEMS

Rabih ZAKARIA, Laurent MOALIC, Alexandre CAMINADA

OPERA, UTBM
Belfort, France
{rabih.zakaria, laurent.moalic,
alexandre.caminada}@utbm.fr

Mohammad DIB

GDF SUEZ - CEEME
Paris, France
mohammad.dib@gmail.com

ABSTRACT: *In one-way carsharing systems, users can take a car from a station and return it to any other station. Available cars and free parking space at each station, play a major role in the success of these systems. Therefore, carsharing operators recruit employees to relocate cars between the stations. This is done to avoid rejection of users' demands for taking cars from the station or returning them. In this paper, we present a greedy algorithm in order to reduce the number of rejected users' demands, using the minimum number of employees. We used mobility data collected from an operational system to build the users' demand matrices as input for our algorithm. For example, in a carsharing system which has 20 stations, 150 cars, through more than 1,370 trips made during one day and more than 555 expected rejected demands, results show that our algorithm is able to reduce 55% of expected rejected demands using 5 employees and more than 80% of these rejected demands using 10 employees. We compared the algorithm with a Mixed Integer Linear Programming model solved using IBM ILOG CPLEX optimizer and made the proof of performance.*

KEYWORDS: *Transport, Carsharing, Optimization, Approximation algorithm.*

1 INTRODUCTION

Nowadays, cars are at the core of environmental concerns due to their significant contribution to the environmental problems that affect our globe. Most of the major cities worldwide suffer from congestion, vehicles emissions, lack of parking spaces and noise pollution. Over the last few years, cities have started to think differently about private car usage and have provided new solutions that reduce car ownership while simultaneously offering private car advantages. For instance, the concept of carsharing service is one of these solutions that can contribute positively to solve these problems.

There are different ways to share vehicles such as company and neighborhood carsharing, station cars, and carsharing with multiple stations (Barth and Shaheen, 2002). In the first two models, trips are mostly round-trips whereas in the third one, they are almost one-way. One-way trips lead to additional cost for the system operator because they require recruiting staff to relocate vehicles periodically between the stations (Kek *et al.*, 2009). This is done to maintain a minimum number of vehicles in each station in order to satisfy clients' demands. Consequently, it is necessary to avoid the instances of overfull stations that do not have any available parking space to users who want to return a car; also to avoid empty station instances that do not have any available vehicles for use. In this kind of systems, the demand fluctuates throughout the day. Hence, the system may become imbalanced: some stations become empty so the client's request to take a car will be rejected. At the same time, other stations may become full, so the client's request to return a car will be rejected as well. Therefore, the client has to wait or look for another station in the neighborhood to get or return a car. To solve this imbalance problem, carsharing operators have to recruit staff to move cars between the stations to meet the clients' demands; in the following, we refer to this staff by "jockeys". Jockeys will have to relocate cars between stations. In other words, they will bring cars to the empty stations and remove cars from the full stations. Obviously, relocating cars would be ideal if it alleviates two rejected demands at the same time. However, the way to do it is not so simple, because it depends on the car rental demand along the day and the distance and the time needed to move between the full and the empty stations. Moving a car from a full station to a station with free parking spaces may be also a good compromise solution to increase the car's rental demand satisfaction and to limit the jockey cost. 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.

There are many related works in the literature dealing with this problem, both for car and Bike sharing. (Barth, Todd and Xue, 2004) proposed to use the client himself to contribute in the relocation operation; although this approach was successful in reducing 42% of the overall number of relocation operations, this only works when 100% of clients participate, which is obviously not always guaranteed. (Cheu *et al.*, 2006) compared two-trip forecasting models, namely neural networks and support vector machines, in a multiple-station shared-use vehicle

system, but they did not deal with the cars relocation problem. Whilst (Kek *et al.*, 2006) developed a simulation model that helps multiple-station shared-use vehicle operators to recognize the efficient resources' combination and system set up according to two selected relocation techniques: shortest time and inventory balancing. 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, (Kek *et al.*, 2009) presented a decision support system for car sharing companies to determine a set of staff and operating parameters for the car relocation problem. Tested on a set of commercial network from a car sharing 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.

In next sections, we will first present physically and formally the problem we tackle, the data we used for the study, and then the description of the optimization algorithm and the results based on simulation.

2 RELOCATION PROBLEM

2.1 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 client's 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 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 rejected demands at different times.

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 S_i, 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 S_i 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 places 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.

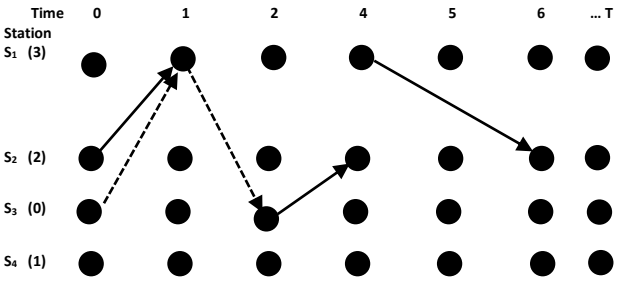

Figure 1: Simple representation of a carsharing system in an urban area.

Thus, to reduce a rejected demand, 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.

2.2 Mixed Integer Linear Programming formulation for the relocation problem

Starting from (Kek *et al.*, 2009), 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 ma-

trix 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 $A_2\{..., a_2(i_t, j_{t+tij}), ...\}$. 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 $A_3\{..., a_3(i_t, j_{t+tij}), ...\}$. 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 a Mixed 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_{it} : Integer variable to represent the number of rejected demand to take a car out of a station *i* at time step *t*.

in^r_{it} : Integer variable to represent that number of rejected demand to return a car into a station *i* at time step *t*.

On 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_{it} : Represents the number of demands to return a car into a station *i* at time step *t*.

p_i : Number of parking places 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_{i_t} : Number of available vehicles at station *i* at time step *t*.

The MILP model for the problem is:

$$Min \quad Z = c_{ij}\Big(\sum_{(i_t, j_{t+t_{ij}})\in A3}\sum_{e\in E} move^e_{i_t j_{t+t_{ij}}} + \sum_{(i_t, j_{t+t_{ij}})\in A4}\sum_{e\in E} rel^e_{i_t j_{t+t_{ij}}}\Big) + c_{out}\sum_{i_t\in V} out^r_{it} + c_{in}\sum_{i_t\in V} in^r_{it} + c_e\sum_{e\in E} u^e \tag{1}$$

$$Subject\ to:$$

$$\sum_{i\in S} wait^e_{i_1 i_2} + \sum_{\substack{i,j\in S\\ i\neq j}} move^e_{i_t j_{t+t_{ij}}} + \sum_{\substack{i,j\in S\\ i\neq j}} rel^e_{i_t j_{t+t_{ij}}} = u^e \quad \forall e \in E \tag{2}$$

$$wait^e_{i_{t-1} i_t} + \sum_{(j_{t-t_{ij}}, i_t)\in A3} move^e_{j_{t-t_{ij}} i_t} + \sum_{(j_{t-t_{ij}}, i_t)\in A4} rel^e_{j_{t-t_{ij}} i_t} - wait^e_{i_t + i_{t+1}} + \sum_{(i_t, j_{t+t_{ij}})\in A3} move^e_{i_t j_{t+t_{ij}}} + \sum_{(i_t, j_{t+t_{ij}})\in A4} rel^e_{i_t j_{t+t_{ij}}} = 0 \quad \forall i_t \in V, e \in E, t > 1 \tag{3}$$

$$av_{it} = av_{it-1} + (in_{it} - in^r_{it}) - (out_{it} - out^r_{it}) + \sum_{(j_{t-t_{ij}}, i_t)\in A4}\sum_{e\in E} rel^e_{j_{t-t_{ij}} i_t} - \sum_{(i_t, j_{t+t_{ij}})\in A4}\sum_{e\in E} rel^e_{i_t j_{t+t_{ij}}} \quad \forall i_t \in V \tag{4}$$

$$av_{it} \le p_i \quad \forall i_t \in V \tag{5}$$

$$in^r_{it} \le in_{it} \ \forall i_t \in V \tag{6}$$

$$out^r_{it} \le out_{it} \ \forall i_t \in V \tag{7}$$

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

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

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

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

$$in^r_{it} \ge 0 \quad \forall i_t \in V \tag{12}$$

$$out^r_{it} \ge 0 \ \ \forall i_t \in V \tag{13}$$

$$av_{it} \ge 0 \ \ \forall i_t \in V \tag{14}$$

The objective function (1) minimize the number of rejected demands and the number of relocation operations needed to reduce the 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. This depends on 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 positive.

3 MOBILITY DATA

In this section, we will describe the mobility data that we used in our study to generate the inputs for our algorithm.

To build the mobility data, we used two main types of information:

- GIS shape files describing the geographical entities
- Survey data and socio-economical information collected by professionals for regional planning needs, describing the main flows of people mobility.

Collected data concerns a 20 km * 10 km area in a region of Paris, France. More details can be found in (Moalic et al., 2013). The area is divided into a grid of equal cell size. Each cell is characterized by two properties:

- Terrain type: static information representing the dominant structure type of the area covered by the cell (roads, buildings, houses, company locations, etc.)
- Attraction weight: dynamic information that varies every 15 minutes during the day. Attraction weights are computed according to cells terrain type and all other survey data.

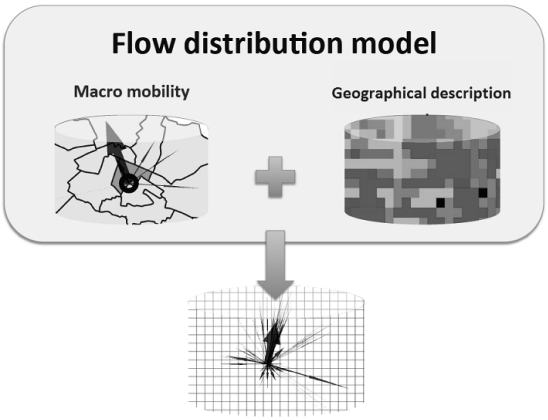

Figure 2 Flows distribution in mesh

This mobility data is used to build the people mobility between each cell during the day. All the flows are set in 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. Figure 2 shows the outflows from the selected cell, obtained from macro data and geographical description. The thickness of the arrows reflects the number of people moving between cells. For the purpose of this study, we define a cell's size to be 300 m. Since some cells cannot be an origin or destination of a mobility flow, such as fields or lakes, for our experimentation, nearly 400 cells are considered. That gives 160,000 origin/destination couples. Considering time steps of 15 minutes each (96 per day), the flow matrix used contains more than 15,000,000 records describing how people are moving during the day.

3.1 Stations location

In this section, we will describe how we located the carsharing stations in our zone of study in a way that they cover the maximum demand in an appropriate manner. In our mobility data, stations are located using a

multi-objective local search algorithm. In this algorithm, we aim to optimize three objectives:
f_1: flow maximization i.e. the locations must allow us to maximize the flows between themselves.
f_2: balance maximization i.e. the location must allow us to maximize the balance between inflows and outflows of a station.
f_3: minimization of flow standard deviation i.e. the location must allow us to get a uniform flow along the day.

3.2 Forecasting who will use the service

The algorithm for locating the stations and the evaluation of users' demand during each time step, suppose a good understanding of who will use the service. This part was realized 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.

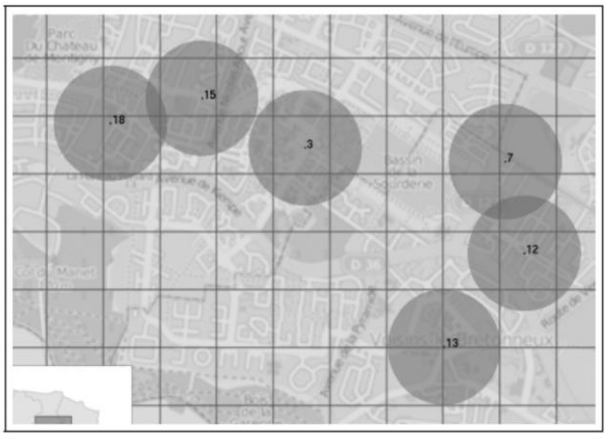

Figure 3 Stations and cells coverage

Figure 3 shows stations and their coverage area (disc). The inflows / outflows associated to each station are the weighted sum of the flows from the covered cells. The weight depends on the proportion of coverage. If several stations cover the same part of a cell, the associated flows are divided between all of them.

Using the mobility data, we were able to locate 20 stations in the region where we did our study; we assumed that each station contains 10 stalls. 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 MILP described earlier in this paper. When we set the number of employees to zero, constraint (4) will be simplified to take the form below:

$$av_{i_t} = av_{i_{t-1}} + (in_{i_t} - in^r_{i_t}) - (out_{i_t} - out^r_{i_t}) \quad (4a)$$

Using this new constraint and the other boring constraints, 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_{it} : 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. In the following, we consider a carsharing system that has 20 stations, 150 cars and more than 1370 trips during 24 hours of a day.

4 JOCKEYING ALGORITHMS

4.1 Solving the relocation problem using CPLEX solver.

We solved our MILP model using IBM CPLEX solver on an Intel core I5 machine of 16 GB RAM. Although the MILP model gives us an optimal solution for the relocation problem, the time needed to solve the problem, increases dramatically when we add the

number of employees involved in the relocation processes. Figure 4 shows the time needed (minutes) to solve the MILP relocation problem when we increase the number of jockeys. At the beginning, when the number of jockeys involved in the relocation operation is small, the resolution of the MILP model using CPLEX takes less than one minute. However, the execution time increases dramatically when the number of jockeys exceeds five where it takes more than one hour to solve the MILP model. After that, when the number of jockeys exceed thirteen, the execution time exceeded three hours to give the optimal solution for our relocation problem.

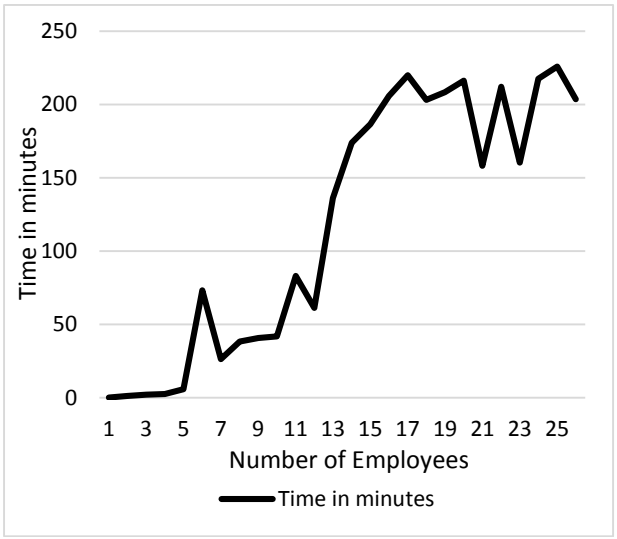

Figure 4 Solving time (minutes) of the MILP model using CPLEX when varying the number of employees

4.2 A Greedy Algorithm to solve the relocation problem.

In order to solve the relocation problem in a faster way, we developed a greedy algorithm to reduce the rejected demands. The algorithm tries to alleviate 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 alleviate 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 reduce as much rejected demand as possible. The best the jockey can do is to alleviate 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 reduces 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 alleviate this rejected demand and in the same time will try to anticipate and avoid 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 avoid 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 reduce them as it is indicated in the flowchart in figure 5.

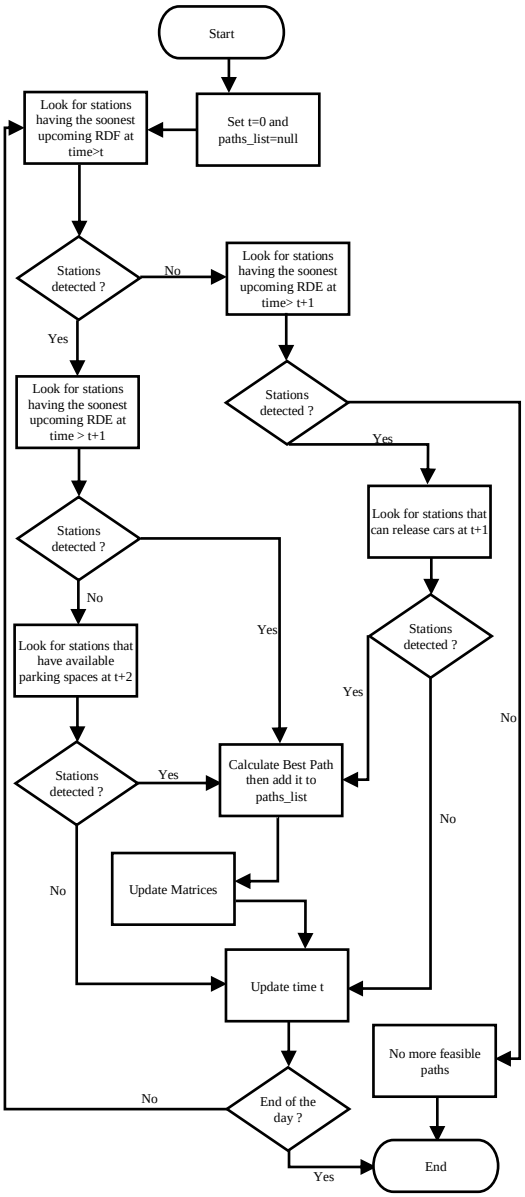

Figure 5 Flowchart of the Relocation Algorithm

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:

- Starting station where the jockey starts the relocation operation at relocation starting time.
- Intermediate station where the jockey arrives to pick up a car at relocation intermediate time.
- 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. The path starts from one origin station (from the station where the jockey is). However, the jockey has different possibilities to choose between many intermediate and destination stations. A jockey should consider the path of minimum cost, which is the one that reduce 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 reduce the maximum rejected demands in a way that we privilege the paths that alleviate the soonest upcoming rejected demands. To calculate the minimum path cost we use this formula:

$$cost = \frac{1}{E - G + 1} + \Delta \tag{15}$$

Where:

$$\Delta = \frac{1}{t_G + 1} \times G - \frac{1}{t_E + 1} \times E \tag{16}$$

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.

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 reduce 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 alleviate 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 reduce 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 alleviate 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 stalls.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.

5 RESULTS AND EXPERIMENTATION

We did a comparison between the performances of the two approaches we used in our study. The execution

time of our greedy algorithm is less than one second while that of MILP reaches more than 3 hours when we increase the number of jockeys to 15 or more, as we described earlier in this paper.

5.1 Greedy Algorithm VS Exact solver results

Before executing the algorithm, we had 59/96 time steps having rejected demands. Therefore, if we are using just one jockey, in the best cases, he can reduce 118 rejected demands (59 x 2), 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 reduced rejected demands will decrease when the number of time steps having rejected demands decreases. The graph in figure 6 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 solved by the MILP model. Initially, before execution of the algorithm we had 556 rejected demands. After execution of our greedy algorithm, the results show that when we use 10 jockeys, we could reduce 81% of the rejected demands, while our MILP model reduced nearly 82.5%. These results reveal the good performance of our greedy algorithm.

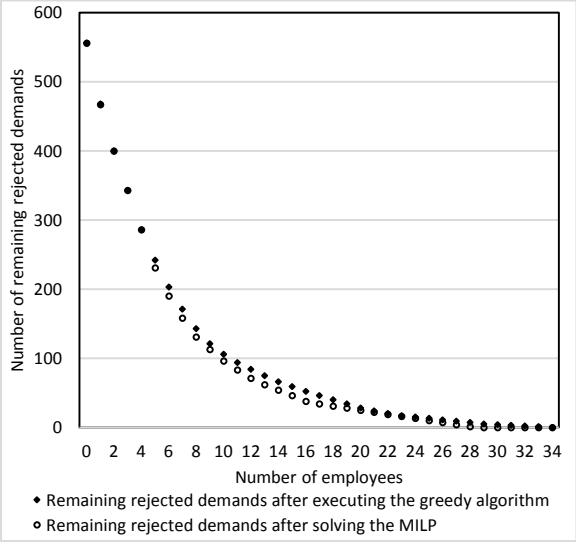

Figure 6 MILP VS our Greedy Algorithm results

CONCLUSION AND FUTURE WORKS

As aforementioned, in this study, we started by presenting the relocation problem for a carsharing system. Then, we formulated the problem as a Mixed 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 reduce the maximum number of rejected demands in a minimum path cost either by re-solving the rejected demands directly or by making anticipated car movements that reduce rejected demands before their occurrence. The greedy algorithm showed impressive performance in comparison to the MILP model. It finds competitive solutions in less than one second while the exact MILP solver takes more than one day when we use a large dataset and a big number of jockeys. After running our algorithm on a set of data, the results show that, for a carsharing system that has 20 stations and 150 cars: 10 jockeys were able to alleviate more than 80% of rejected demands.

In future work, we will model a sophisticated 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. We will also study the effect of stochastic data on the performance of the two approaches describer earlier in this paper to evaluate the efficiency of the offline route planning in situations similar to real life where stochastic data prevails.

REFERENCES

Barth, M., & Shaheen, S. A. (2002). Shared-use vehicle systems: Framework for classifying carsharing, station cars, and combined approaches. *Transportation Research Record: Journal of the Transportation Research Board*, 1791(1), 105-112.

Barth, M., Todd, M., & Xue, L. (2004). User-based vehicle relocation techniques for multiple-station shared-use vehicle systems.

Cheu, R. L., Xu, J., Kek, A. G., Lim, W. P., & Chen, W. L. (2006). Forecasting shared-use vehicle trips with neural networks and support vector machines. *Transportation Research Record: Journal of the Transportation Research Board*, 1968(1), 40-46.

Kek, A. G., Cheu, R. L., Meng, Q., & Fung, C. H. (2009). A decision support system for vehicle relocation operations in carsharing systems. *Transportation Research Part E: Logistics and Transportation Review*, 45(1), 149-158.

Kek, A. G., Cheu, R. L., & Chor, M. L. (2006). Relocation simulation model for multiple-station shared-use vehicle systems. *Transportation Research Record: Journal of the Transportation Research Board*, 1986(1), 81-88.

Moalic, L., Lamrous, S., & Caminada, A. (2013). A Multiobjective Memetic Algorithm for Solving the Carsharing Problem. *Proceedings Of The 2013 International Conference On Artificial Intelligence-Icai 2013*, Vol. 1, pp. 877-883.