Multiobjective car relocation problem in one-way carsharing system

Rabih Zakaria¹ · Mohammad Dib² · Laurent Moalic³

Received: 4 July 2018 / Revised: 3 November 2018 / Accepted: 8 November 2018 / Published online: 27 November 2018
© The Author(s) 2018

Abstract In this paper, we present a multiobjective approach for solving the one-way car relocation problem. We fix three objectives that include the number of remaining rejected demands, the number of jockeys used for the relocation operations, and the total time used by these jockeys. For this sake, we propose to apply two algorithms namely NSGA-II and an adapted memetic algorithm (MA) that we call MARPOCS which stands for memetic algorithm for the one-way carsharing system. The NSGA-II is used as a reference to compare the performance of MARPOCS. The comparison of the approximation sets obtained by both algorithms shows that the hybrid algorithm outperforms the classical NSGA-II and so solutions generated by the MARPOCS are much better than the solutions generated by NSGA-II. This observation is proved by the comparison of different quality indicators' values that are used to compare the performance of each algorithm. Results show that the MARPOCS is promising to generate very good solutions for the multiobjective car relocation problem in one-way carsharing system. It shows a good performance in exploring the search space and in finding solution with very good fitness values.

Keywords Carsharing · Car relocation · Integer linear programming (ILP) · Multiobjective optimization · Memetic algorithm · NSGA-II

1 Introduction

Carsharing is a new concept for using a car by sharing it between multiple users. This kind of systems turns the mobility into a service that allows its users to enjoy the advantages of owning a private car without the need to buy it. Carsharing operators maintain a fleet of cars that are available for use to their members who usually pay yearly registration fees, a monthly membership and are billed on the usage according to time spent and distance traveled.

What makes the carsharing more convenient is that members do not have to care about insurance, maintenance and even fuel price, which are included in the usage fees and shared between all members. Usually carsharing companies provide a short-term car rental. Members can have access to vehicles on the go or according to a reservation in advance via telephone, smartphones or website; cars are available 24 h a day, 7/7. The fleet of cars is usually spread in a way that every potential member (user) can have a car in his neighborhood at a convenient walkable distance.

There are multiple models of carsharing systems. For instance, the free-floating model allows users to pick up and return their rented car to any legal parking space within a defined area. This kind of system is convenient to users, since they do not have to stick to defined stations to park or to pick up a car. Unlike free-floating carsharing systems, the round-trip model is a station-based carsharing system where the user has to return the car to the station of departure. Finally, one-way carsharing provides an

✉ Mohammad Dib
mohammad.dib@gmail.com

Rabih Zakaria
rabih.zakaria@ulf.edu.lb

Laurent Moalic
laurent.moalic@uha.com

¹ Université de Technologie et de Sciences Appliquées Libano-Française, Tripoli, Lebanon

² ENGIE, 92930 Paris, France

³ UHA, Mulhouse, France

interesting compromise between two previous approaches. It is another model of station-based systems where a user can drive a car from a station and ends his reservation at any other station in the system.

In our paper, we focus on one-way carsharing system. The one-way flexibility makes the system more attractive to its users. However, this has a major drawback. It often leads to an imbalance in the number of cars in stations. On one side, some stations will reach their maximum parking capacity, forcing users who want to return their car in these stations to look around for other stations that have available parking places. On the other side, other stations will not have any available car for a new arriving user who wants to take a car from these stations. When the imbalance problem occurs often, the carsharing system becomes less attractive since it is unreliable for users. Eventually, it may lead to service failure.

In this context, carsharing operators are working hard to alleviate this problem. For this sake, carsharing companies recruit employees to redistribute cars on the stations based on the users demand for cars. In this paper, we refer to these employees by jockeys and the redistribution operations by relocation operations. Car relocation operations can improve the system reliability but in the same time, they increase the cost of operating the system. Here emerges the need for optimizing these operations in order to reduce the cost of relocating the cars and to increase the system reliability. A relocation operation consists of moving a car from a station that has unused cars or from a station that needs free parking spaces for near future demands, to another station that has a shortage in the number of available cars for users who want to start their trip starting from this station.

It is good to know that bikesharing systems have the same mobility concept and eventually have the same relocation problem. However, the size of a bike is much smaller than the size of a car. Consequently, the bike-sharing operator can relocate many bikes using a truck, which is not possible in the case of cars. In this paper, we consider that a jockey relocates one car at a time.

Many research papers have treated the car relocation problem. In [1], a study is presented for the relocation problem in one-way electric vehicle sharing system in Praxitele, France. The authors suggested a fleet of finite capacity to redistribute cars over the stations. The problem was modeled as pickup and delivery and formulated as a mixed-integer programming. After that, Barth and Todd [2] developed a queuing-based discrete event simulation model to analyze the performance of multiple station carsharing system with one-way trips. In 2006, Kek et al. [3] developed a time-stepped simulation model to evaluate the performance of one-way carsharing system. In order to define the best combination of system parameters, Kek

et al. [4] proposed a new approach that is based on the simulation model developed in [3]. The set of recommended parameters suggested by the three-phase decision support system could lead to a 50% reduction in the staff cost, an improvement in the zero-vehicle time (empty stations) and full-port time (full stations) and decrease in the number of relocations between 37.1% and 41.1%. In another study, Cucu et al. [5] proposed another approach to help decision makers to maintain a balanced number of cars in each station and to evaluate the potential of new locations for carsharing stations. The authors developed a fuzzy logic algorithm that is based on the preferences of potential customers (departure time, weather conditions, day of the week, and traffic conditions) to forecast clients demands. The authors concluded that a good estimation of client demands is crucial for improving the level of service and relocation operations of a carsharing system. Nair and Miller-Hooks [6] developed a stochastic, mixed-integer linear programming model with a joint chance constraint in order to plan the vehicle relocation operations. This model aims to minimize the cost of these operations and to guarantee the satisfaction of a proportion of all near-term demand scenarios. The study showed that the proposed relocation operations that consider the demand uncertainty could highly increase the level of service of the carsharing system. Smith et al. [7] presented a study for the problem of relocating the employees in charge of the relocation operations in a one-way carsharing system. The study that is based on a fluid model focused on the optimization of the relocation operations in order to reduce the number of relocation vehicles and the employees who drive these vehicles. The simulations suggested that the number of relocation employees required for the relocation operations should be between 1/3 and 1/4 of the number of the vehicles in Euclidean network topologies. It is good to mention that many papers studied the case where the client himself is used to relocate vehicles by offering them incentives such as additional free time or bonus credits [8] or cheaper dynamic prices for their trips [9–12]. Although client participation in the relocation operations can contribute in reducing the number of required relocation operations, this is often not guaranteed [13, 14]. In more recent papers, Boyaci et al. [15] tackled the problem of the optimization of vehicle and personnel relocation operations in one-way electric carsharing systems with reservations. They used a multiobjective mixed-integer linear programming optimization and a discrete event simulation framework. Their results showed that efficient algorithms play a major role in relocation operations. Also, Huang et al. [16] presented a mixed-integer nonlinear programming model (MINLP) to locate stations and determine their capacity for one-way carsharing systems taking into account the car relocation operations with flexible travel demand. A

gradient algorithm is used to solve the MINLP. When applying the model on a real-world system in China, results showed that the algorithm can be used to solve real-world large-scale problem. Moreover, Xu et al. [17] developed a mixed-integer nonlinear nonconvex model for the electric vehicle fleet size for one-way carsharing systems taking into account the vehicle relocation and personnel assignment. The model was tested on a one-way carsharing system in Singapore to prove its efficiency. In this paper, we first present the transition from mono to multiobjective formulation of the studied problem. Then, we propose a multiobjective formulation for solving this problem. After that, we present the resolution methods that we use to solve the car relocation problem. Next, results are shown and finally a conclusion and perspective are provided. To our knowledge, papers that deal with the multiobjective aspect of the car relocation problem in one-way carsharing system are really scarce.

2 One-way carsharing: a multiobjective formulation

2.1 From mono to multiobjective

A typical formulation of the car relocation problem in one-way carsharing system with its Integer linear programming (ILP) model and related constraints can be found in [18]. Below, presented the related objective function (1):

$$\begin{aligned} \text{Min } Z = & cMove \left(\sum_{(i_t, j_{t+t_{ij}}) \in MA} \sum_{e \in E} move_{i_t j_{t+t_{ij}}}^e \right. \\ & + \left. \sum_{(i_t, j_{t+t_{ij}}) \in RA} \sum_{e \in E} rel_{i_t j_{t+t_{ij}}}^e \right) \\ & + cRejectEmpty \sum_{i_t \in V} outR_{i_t} \\ & + cRejectFull \sum_{i_t \in V} inR_{i_t} \\ & + cStaff \sum_{e \in E} u^e. \end{aligned} \quad (1)$$

Several decision variables are used in (1) :

- $move_{i_t j_{t+t_{ij}}}^e$: When an employee e is involved in a moving activity, the associated binary variable is assigned with the value 1, while it remains 0 otherwise.
- $rel_{i_t j_{t+t_{ij}}}^e$: When an employee e is involved in one of the relocation activities, the associated binary variable is assigned the value 1, while it remains 0 otherwise.

- u^e : When an employee e is used at least once during the day, its associated binary variable takes the value 1, while it takes the value 0 otherwise.
- $outR_{i_t}$: This variable can be assigned with integer values. It represents the number of rejected user demands to rent a car from station i at time t .
- inR_{i_t} : This variable can be assigned with integer values. It represents the number of rejected user demands to give back the rented car to a station i at time t .

On other side, the input parameters used in the objective function (1) are listed below:

- $cMove$: Denotes the estimated cost of a relocation or moving activity from a station to another one.
- $cRejectEmpty$: Stands for the estimated cost of the rejection of a demand to rent a vehicle from a station.
- $cRejectFull$: Stands for the estimated cost of the rejection of a demand to give back a vehicle to a station.
- $cStaff$: Denotes the estimated cost of an employee during a day.

In such approach, the objective function (1) minimizes the weighted aggregation of the number of rejected demands to take or to return a car, the number of employees and the number of move and relocation operations needed to reduce the number of rejected demands. Related constraints are used to make sure that each employee is involved in only one activity at a time, while others are used to make sure that an employee will not start a new activity until he finishes the previous one and to guarantee a continuity for the employee's activities. In addition, some constraints are used to calculate the number of available vehicles at each station at each time step and to make sure that the number of available vehicles at a station cannot be greater than the capacity of the station. And finally, there are constraints used to ensure that the number of rejected demands at a station cannot exceed the demand itself. This problem is addressed in an exact way in [3] on small instances. For bigger instances, a greedy algorithm [19] has proved its performance regarding the execution time and the quality of the generated solutions in a comparison with an exact solver (CPLEX). As a result, these methods return a planning for the operations that a jockey should perform in order to reduce the number of rejected demands. A relocation planning consists of a schedule for the car relocation operations that should be done during the day. Two main drawbacks can be noticed regarding this formulation:

- First, when we solve the car relocation problem using the two approaches described earlier, we consider that a jockey is working all the day, regardless of the number

of operations he is doing. However, while analyzing the planning, we notice that the schedule for some jockeys is scattered all over the day. For example, even if a jockey has to do two operations only, one at the beginning of the day and another one at the end of the day, the jockey will be counted as any other jockey that has a schedule full of relocation operations during the whole day. For that reason, when we present the solutions provided by the two developed approaches for the decision makers, we are missing one important aspect of the problem, which is the total time spent by jockeys for the relocation operations plan. Adding a new constraint for the integer linear programming model makes the problem harder and getting results for real dataset using an exact solver in reasonable time is not feasible currently.

- Second, the mono-objective approach considers in the same objective function, the cost of a jockey ($cStaff$) and the cost of a rejected demand ($cRejectFull$ and $cRejectEmpty$); this latter which can be seen as the carsharing price. Hence, it supposes to be able to fix these costs. Despite the importance of the cost parameters, in [18] those costs were left behind for simplicity reason. Setting different cost values will change the entire results of an ILP since we have one objective to optimize. But it should be noted that while $cStaff$ is somehow easy to determine, the cost of a rejected demand is very difficult to evaluate. We can calculate the loss of not satisfying a demand; however, we also have to calculate the negative impact on the image of the service and eventually the risk of not using the service in the future. For these two reasons, using a multiobjective approach can be seen as a powerful alternative to the existing algorithm. In this approach, each objective is treated separately and independently from other objectives. Thus, it is not necessary to take costs into consideration when they are the same for each rejected demand, for each relocation operation and for each jockey. However, if costs were not the same (from one trip to another, from a relocation operation to another, from a jockey to another), it will be necessary to integrate them in the objective functions (for example, weighted objective functions, with non-constant costs, may lead to obtain solutions that prioritize solving rejected demands having high carsharing prices).

2.2 Integer linear programming formulation

The relocation problem can be modeled as a two-dimensional time-space matrix of size $N \times M$, where N is the total number of stations $S = \{1, 2, \dots, N\}$ and M is the

number of time steps $T = \{1, 2, \dots, M\}$ in the day. Each element of the matrix represents a station s_i at time t . For each station $s \in S$ we generate M nodes to represent that station at each time t . Then we put all the $S \times T$ nodes in one set $V = \{1_1, \dots, 1_T, \dots, N_1, \dots, N_M\}$. During the day, we consider that an employee is always involved in one of these three types of activities:

1. Relocating: is the action taken by the jockey to move a car from a station i to another station j .
2. Moving: is the action taken by the jockey to move himself from his current station to another station in order to begin a relocation activity.
3. Waiting: when the jockey is not involved in relocating or moving activities we say that the jockey is waiting.

Therefore, to represent these activities we generate three sets of arcs in the time-space network. For each node $i_t \in V$, we construct an arc wa that represents a waiting activity between i_t and i_{t+1} ; we call this set $WA = \{\dots, wa(i_t, i_{t+1}), \dots\}$. Then, for each node i_t in V , we construct $N - 1$ arcs ma 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 name this set $MA = \{\dots, ma(i_t, j_{t+t_{ij}}), \dots\}$. In the same way of moving activities, we build $N - 1$ arcs ra to represent relocation activities for each station, and we denote this set $RA = \{\dots, ra(i_t, j_{t+t_{ij}}), \dots\}$. We represent the available staff that will be involved in doing these activities by a set $E = \{1, \dots, e, \dots, W\}$ where W is the maximum number of available employees. The ILP model described in [18] uses six decision variables, the variables related to the present paper here are:

- u^e : Binary variable, that takes the value 1 if the employee e is used during the day and 0 otherwise.
- $rel_{i,j,t+t_{ij}}^e$: Binary variable associated with the set of relocation activities RA . It takes the value 1 if employee e has been relocating a car from station i to station j , from time step t to $t + t_{ij}$ and 0 otherwise.
- $outR_{i_t}$: Integer variable to represent the number of rejected demand to take a car out of a station i at time step t .
- inR_{i_t} : Integer variable to represent the number of rejected demand to return a car into a station i at time step t .

The mathematical model for the car relocation problem in one-way carsharing system was solved with a greedy approach in [19]. The proposed greedy search method proved its performance regarding the execution time and the quality of generated solutions in a comparison with an exact solver (CPLEX). However, a simple analysis of the results shows that the proposed method did not treat the

multiobjective aspect of the car relocation problem. In the greedy search algorithm, the objective was to minimize the number of rejected demands. However, it is important to integrate other objectives. In fact, the proposed methods return a planning for the operations that a jockey should perform in order to reduce the number of rejected demands. A relocation planning consists of a schedule for the car relocation operations that should be done during the day. On the other hand, the results provided by the aggregated ILP model do not show the Pareto optimal solutions. To our knowledge, few studies deal with the multiobjective aspect of the car relocation problem in one-way carsharing system. Therefore, we decide to develop a multiobjective optimization algorithm that tackles this problem from this perspective. In the following, we present a multiobjective formulation for the relocation problem in one-way carsharing system.

2.3 Objective functions definition

As described earlier, carsharing operators recruit employees to relocate cars between stations in order to reduce the number of rejected demands for taking a car or returning it to a station. Each car relocation operation $op(i, j)$ starts when a jockey goes from his current location to station i where he will take a car; then, it ends at a destination station j where the jockey will deliver that car, with $i \neq j$. The duration of each relocation operation depends on the time needed to move between the stations. For this sake, we have a time matrix that contains the time needed to move between each pair of stations during the day based on the distance between these stations. For a carsharing system that has N stations we have $N \times (N - 1)$ relocation possibilities (since moving a car from a station i to the same station i does not make sense). Then, we put all the possible combinations of operations in one array that we name $OP = (\dots, op_k(i, j), \dots)$. Each element of OP has an index k that refers to one relocation operation, $k \in [1, N \times (N - 1)]$. Relocation operations can take place at different times of the day.

In this section, we present a multiobjective optimization algorithm where we consider three objectives to minimize:

2.3.1 Minimize the total number of remaining rejected demands: f_1

As described earlier, the relocation operations are necessary to reduce the number of rejected demands. The impact of each relocation operation a_k at time t can be measured by the number of reduced or generated rejected demands $\Delta_{a_{k_t}}, \Delta_{a_{k_t}} \in [-2, +2]$. In the best case, $\Delta_{a_{k_t}}$ can reduce two rejected demands (-2 rejected demands). However, $\Delta_{a_{k_t}}$

can generate two rejected demands ($+2$ rejected demands) when we make a bad relocation decision.

A relocation operation consists of moving a car from a station i and relocates it to another station j . If for example the employee driver moved a car from a station i that had just one car, and few seconds later, a client arrives at this station to rent a car, he will generate a new rejected demand ($+1$). Within the same example, if the employee driver moved a car to station j that had $n - 1$ cars where n is the maximum number of cars a station can hold. And few seconds later, a client has arrived to this station to return his car, and he will not find any available parking space and thus another rejected demand will occur ($+1$). So that relocation operation will generate two rejected demands ($+1 + 1 = +2$). On the other side, if the employee driver moved a car from a station i that had n cars, and few seconds later a client arrived to return his car, he will find an available parking space, thanks to the employed driver because he reduced one rejected demand (-1). In this same relocation operation, the employed driver moved the car to station j that had zero cars, and few seconds later, a client arrived to rent a car, the client will find a car to drive, thanks also to the employed driver because he also reduced one more rejected demand (-1). So this relocation operation reduced 2 rejected demands ($-1 - 1 = -2$). And sometimes the relocation operation will not have any direct effect on the rejected demands, so it will reduce 0 rejected demands.

To calculate the total number of remaining rejected demands f_1 , we do the summation of the impact of each relocation operation $\Delta_{a_{k_t}}$ in each list of relocation operations L_t , then we add the initial number of rejected demands R_i

$$f_1 = R_i + \sum_{L_t \in P} \sum_{a_{k_t} \in L_t} \Delta_{a_{k_t}} \quad t \in TS. \quad (2)$$

Since the objective of the relocation operations is to reduce the number of rejected demands, we make sure to remove any operation that generates additional rejected demands. Thus, we only keep operations that reduce the value of f_1 .

To relate to the ILP model described in Sect. 2.2, f_1 can be expressed as follows:

$$f_1 = \left(\sum_{i_t \in V} outR_{i_t} + \sum_{i_t \in V} inR_{i_t} \right). \quad (3)$$

2.3.2 Minimize the number of jockeys involved in the relocation operations: f_2

To calculate the number of jockeys needed to reduce the rejected demands, we create an array of integers U to count the number of jockeys involved in relocation operations

during each time step. We refer to this array by $U = u_0, \dots, u_t, \dots, u_T$. As aforementioned, a jockey starts a relocation operation at the departure time t and finishes it at the arrival time to the destination station at time $t + t_{ij}$ where t_{ij} is the time needed to move from station i to station j . For each accomplished relocation operation, we increase the correspondent u_t values in the range $[t, t + t_{ij}]$. For example, if a jockey starts a relocation operation at time $t = 1$ and he needs two time steps to arrive to the destination station (t_{ij}), then we should increment the correspondent two elements of the array U . However, for reasons of simplicity we also increment one more element to count the time needed to move from the destination station of the current operation to the next departure station of the next operation (we fixed this value to one). When we get the updated vector U of needed jockeys at each time step, we can get the value of the second objective function f_2 by calculating the maximum value in the array U :

$$f_2 = \max_t(u_t) \quad \forall u_t \in U. \quad (4)$$

To relate to the ILP model described in Sect. 2.2, f_2 can be expressed as follows:

$$f_2 = \sum_{e \in E} u^e. \quad (5)$$

2.3.3 Minimize the total working time of jockeys: f_3

We propose the algorithm 1 to calculate the total operation time of jockeys using the array U described earlier. It starts by the initialization of the used variables. Then, in each iteration, the algorithm calculates the operation time for a jockey and adds it to the total time. We consider that the working time of a jockey j starts when he begins his first operation $t_{u_{\min}}$ of the day and ends at the arrival time of his last performed operation $t_{u_{\max}}$. In the proposed algorithm 1, we start by looking for the first index of the first and the last operation of the day in the U vector that contains the number of operations that should be done at each time step. The difference between these two values represents the working time for a jockey. After that, we decrement the elements in U vector by 1 if $u_t > 0$, starting from the index of the first operation until the index of the last operation. To calculate the working time w_j of a jockey j , we use the formula:

$$w_j = t_{u_{\max}} - t_{u_{\min}} + 1. \quad (6)$$

We calculate the total working time of jockeys by doing the summation of all jockeys working time:

$$f_3 = \sum_{j \in [1; f_2]} w_j. \quad (7)$$

To relate to the ILP model described in Sect. 2.2, f_3 can be expressed as follows:

$$f_3 = \sum_{e \in E} (t_{\max} rel_{i,j,t+t_{ij}}^e - t_{\min} rel_{i,j,t+t_{ij}}^e) \quad (8)$$

$$\forall i_t \in V, e \in E, (i_t, j_{t+t_{ij}}) \in RA.$$

Algorithm 1 Calculation of the total working time of jockeys

Require: U /* U is a vector that contains the number of jockeys needed for the relocation operations during each time step */

```

1:  $Time \leftarrow 0$  /*Set total time to 0 */
2:  $t_{u_{\min}} = 0$  /*Set initial value for  $t_{u_{\min}}$  */
3: while  $t_{u_{\min}} \neq -1$  do
4:    $t_{u_{\min}} \leftarrow$  index of first occurrence of  $u_t \neq 0$  /* -1 if no value found */
5:    $t_{u_{\max}} \leftarrow$  index of last occurrence of  $u_t \neq 0$  /* -1 if no value found */
6:    $Time \leftarrow Time + (t_{u_{\max}} - t_{u_{\min}}) + 1$ 
7:   for  $i = t_{u_{\min}}$  to  $t_{u_{\max}}$  do
8:     if  $U[i] > 0$  then
9:        $U[i] \leftarrow U[i] - 1$ 
10:    end if
11:  end for
12: end while
13: return  $Time$ 

```

In the next two sections, we propose two metaheuristics to solve the multiobjective car relocation problem: the classical NSGA-II and a hybrid memetic algorithm as a hybrid approach. We start by NSGA-II, which will be our reference algorithm to compare the memetic algorithm that will be described after.

3 Resolution methods

In the following, we provide an overview of the methods and approaches used to solve and evaluate the multiobjective optimization problems. After that, we propose two algorithms for solving this problem: the well-known genetic approach NSGA-II and MARPOCS which is a hybrid approach combining genetic and local search approaches.

3.1 Overview of the resolution methods of multiobjective optimization problems

There are multiple approaches proposed in the literature for solving multiobjective problems (MOP). These methods can be classified into three main categories:

- Approaches that are based on the transformation of a MOP into a mono-objective problem. Here we can find methods that are based on the aggregation of the objective functions f_i into one function f . Despite the fact that these methods are easy to implement, they require a good understanding of the studied problem in order to choose the desired values for the coefficient or weight of each objective function.
- Non-Pareto Approaches. In this category, all the objectives are treated separately. The main drawback here is that these approaches find only supported solutions that belong to the convex envelop.
- Pareto approaches: this class of algorithms treats all the objective functions at the same time, taking advantage of the non-dominance concept.

Many studies in the literature prove that Pareto approaches are generally more efficient to solve multiobjective problems. As in mono-objective, hybrid approaches which are based on different simple heuristics, can be also applied in MOP. In many cases, hybrid approaches outperform the pure metaheuristics, as in [20–22].

The two proposed algorithms in the present paper for solving the car relocation problem belong to the Pareto approaches.

These approaches use the dominance relation in the algorithm process. Thus, solutions are not evaluated toward just one objective function; instead, they are evaluated with trade-off solutions that imply all the objective functions. Many Pareto algorithms have arisen after the introduction of the concept of the rank dominance of a solution in the first paper that discussed the Pareto [23]. Since, Pareto concept implies the use of populations, we find many genetic algorithms (GA) that belong to this family. However, Pareto approach also includes local search algorithms that are based on the concept of the Pareto, as well as hybrid algorithms. More recently, indicators-based methods arise in this category. Instead of evaluating the solutions of the population, we evaluate the population itself based on the chosen indicators. In the following, we present a brief description for the ranking operators.

3.2 How multiobjective solutions are compared?

Since a multiobjective solution cannot be evaluated against its objective functions as we do in mono-objective problems, it is important to define a metric capable of representing the quality of a solution. Thus, several methods have been proposed and are often associated with the commonly used GA. The main three methods that are listed in [24] are presented below:

1. *Dominance rank* For a given solution s , the dominance rank is defined by the number of solutions that dominate s , incremented by 1 [25]. Thereby, solutions of rank 1 are non-dominated solutions. As much as the number of solutions that dominate a solution s increases, its rank is worsened. The dominance rank was originally applied in the MOGA method (multiobjective genetic algorithm) [25].
2. *Dominance count* On the contrary of the dominance rank, dominance count is the number of solutions that are dominated by the solution s . A high rank refers to a good solution quality.
3. *Dominance depth* In this approach, solutions are grouped in Pareto fronts [23], non-dominated solutions in the population are assigned the value 1, and then they are removed from the population. The new non-dominated solutions are assigned the value 2, and then they are removed from the population as well. And so on, until that all the solutions of the population are assigned a value. In order to distinguish solutions that belong to the same front, we use complementary mechanisms, as we will see in the next sections. This definition of the dominance depth is used in NSGA algorithms (non-dominated sorting genetic algorithm) [26] and NSGA-II [27] that will be used in this study.

3.3 NSGA-II for the car relocation problem

Although there are many genetic algorithms that can be used in a multiobjective context, NSGA-II is considered a very popular non-dominated sorting genetic algorithm for solving MOP. It is an improved version of NSGA that integrates elitism and with no sharing parameter needed to be chosen *a priori*. It has the advantage of simplicity and good performance. This algorithm is often used as a reference to compare the performance of new approaches. Algorithm 2 recalls the pseudo code for NSGA-II in the general case.

Algorithm 2 Pseudocode for NSGA-II

Require: Population size N , number of generations $nbGen$
*/*a time limit can replace nbGen */*

```

1:  $P \leftarrow \text{init}(N)$  /*Initialize the population P with N random individuals */
2:  $Q \leftarrow \emptyset$  /*Initialize an empty population for children */
3:  $\text{eval}(P)$  /*eval is used for the evaluation of each individual */
4: for  $i=1$  to  $nbGen$  do
5:    $P \leftarrow P \cup Q$ 
6:    $\text{assignRank}(P)$  /* Based on Pareto dominance */
7:   for each front  $f \in P$  do
8:      $\text{setCrowdingDist}(f)$ 
9:   end for
10:   $\text{sort}(P)$  /* By rank and in each rank by the crowding distance */
11:   $P \leftarrow P[0 : N]$ 
12:   $Q \leftarrow \text{buildNextGeneration}(P)$  /* Binary Tournament Selection, Recombination and Mutation */
13: end for

```

In the following, we will describe the genetic operators that we defined in the NSGA-II. Firstly, we present an individual coding, then we describe the selection, crossover and the mutation operators.

3.3.1 Individual coding

A good solution representation is a key factor in developing efficient evolutionary algorithms. The encoding of the solution should enable us to ease crossover and mutation operations, as well as perform easy fitness calculation. Here, a solution s for the car relocation problem consists of a list of relocation operations that are carried out by jockeys during all time steps of the day, $s = (l_1, \dots, l_T)$. Each list of relocation operations l_t contains the indices of the relocation operations that are applied at time t . The size of each list l_t varies between 0 and $R_{\max}$ where $R_{\max}$ is the maximum number of rejected demands for the whole day before starting the relocation operations. A possible representation for car relocation plan may look like as shown in Table 1.

A chromosome is composed of T genes where each gene corresponds to a list of relocation operations l_t . A

Table 1 Representation of a carsharing plan during a day

l_1	(1, 3)	(1, 3)	(1, 2)	
...				
l_t	(10, 5)	(6, 5)	(2, 4)	(3, 7)
...				
l_{T-1}	(2, 6)			
l_T				

relocation operation is represented by its index from the array OP which is the array of all possible combinations of relocation operations. For example, the line l_1 in Table 1 corresponds to relocation operations to do at time step 1, which are moving two cars from stations 1 to 3 and another car from stations 1 to 2.

3.3.2 Selection operators and crossover

In genetic algorithms, we use crossover operator in order to generate a new population by combining individuals from the current population between each other, in an analogy of the biological crossover. When we want to do the crossover, we have to select individuals from our population for the reproduction in the hope of obtaining new chromosome (offspring) that holds the best characteristics of both parents. There are many selection methods such as roulette wheel selection (RWS), elitism selection, rank selection, binary tournament, etc. In NSGA-II, a binary tournament is used: two individuals are chosen randomly from the population then are compared. The best dominant individual is selected to be the first parent. We do the same process for the second parent. Binary tournament is not a very selective method. In that way, NSGA-II avoids the premature convergence.

When we get the pair of parents, we apply the crossover operator. Unlike the selection operator, there is no particular crossover operator defined for NSGA-II. It depends on the studied problem. In our case, we propose a hybrid method for crossover, which is a combination between two points crossover and uniform crossover as shown in Fig. 1.

First, we start by randomly selecting two points for the crossover then we exchange the parts between these two points. After that, we apply the uniform crossover on the remaining parts, which consists of exchanging bits randomly between the two parents. In this way, the proposed crossover operator offers a balance between the uniform crossover that breaks the sequence of operations (high diversification) and the two points crossover that maintains a sequence bloc of operations (weak diversification).

3.3.3 Mutation operator

After the crossover, the mutation operator is used to maintain a genetic diversity in the population by changing one or more genes to new values that we may not find in the parents genes. Usually, mutations occur according to a low mutation probability called mutation rate. In general, we have different methods to apply a mutation depending on the encoding of the solution. For example, in binary encoding, we can apply a simple mutation by inverting the selected bits as shown in Fig. 2.

Fig. 1 Crossover operation

In the proposed model, a special mutation is adapted to the problem representation. For this sake, we choose two random genes from the selected chromosome then, for each selected gene, we apply our dedicated mutation operator, which consists of adding an operation if the gene does not have any operation; otherwise, we randomly select an operation that belongs to this gene and we replace it by another operation selected randomly. When we add or replace a relocation operation, we check if the new operation causes new rejected demands. In this case, we remove that operation which deteriorates each fitness value. That way, the proposed mutation operation can add, replace or remove a relocation operation. We propose to choose two genes for the mutation operator in order to increase the diversification effect, since we find that mutating just one operation is not enough regarding the total number of operations.

3.4 Memetic algorithm

A memetic algorithm (MA) was introduced for the first time by Moscato et al. [28]. According to its definition, a MA consisted of modified version of a genetic algorithm where a local search is integrated. Hybrid approaches have proved their efficiency in solving single objective optimization problems. As a result, researchers were interested in testing the effect of metaheuristics hybridization on the solving MOP. In general, the mechanism consists of adding a local search in a genetic algorithm. The genetic algorithm

Fig. 2 Mutation operation

plays the diversification role. It enables the algorithms to discover more zones in the research space. For every found solution, local search is used to improve its quality and therefore pushing it to the Pareto front.

In a multiobjective context, an algorithm has the role of finding the “best” possible set of non-dominated solutions. In general, Pareto set refers to the feasible non-dominated solutions that cover the entire search space. However, in real-life optimization problems, it is not often easy to obtain all the solutions that belong to the Pareto front. Nevertheless, in most cases we use Pareto front when we want to describe the best found non-dominated solutions.

The general scheme of a memetic algorithm is presented in “Algorithm 3.”

Algorithm 3 General form of a memetic algorithm

Require: N the size of the population, $nbGen$ the number of generations /*A time limit or a any termination condition can be used to replace the $nbGen$ */

- 1: $P_0 \leftarrow \text{init}(N)$ /*Initialize the population P with N random individuals */
- 2: $P'_0 \leftarrow \emptyset$ /*An empty population for children */
- 3: $\text{eval}(P_0)$ /*Evaluate objectives for each individual */
- 4: **for** $i = 1$ to $nbGen$ **do**
- 5: Select P'_i from P_{i-1} based on fitness value
- 6: Apply genetic operator on $P'_i \rightarrow P''_i$
- 7: Apply local search on $P''_i \rightarrow P'''_i$
- 8: $\text{eval}(P'_i, P'''_i)$
- 9: Select P_i from $(P_{i-1} \cup P''_i \cup P'''_i)$
- 10: **end for**

It is important to note that there are different methods to apply a local search in a GA. Therefore, local search can be used with a probabilistic factor just like a mutation, or when the algorithm reaches a given number of generations. Likewise, local search can be applied on every created individual. The choice of how to use the local search and how to maintain an equilibrium between the genetic operator and the local search is crucial to obtain good results.

When designing an algorithm for solving an optimization problem, it is important to study how it processes the search space based on the intensification and diversification aspects. In a genetic algorithm, the crossover and the selection operators are considered as intensification operators while the mutation is used for diversification. However, in a memetic algorithm, roles are exchanged between mutation and crossover: the mutation operator (LS) becomes an intensification operator while the crossover operator becomes a diversification operator. From this perspective, a memetic algorithm can be considered as an improved version of population-based local search where

the crossover operator is used for diversification and the local search is used to ameliorate the found solutions.

3.4.1 Genetic operator in memetic algorithm

There are many genetic approaches that are used in multiobjective context (multiobjective evolutionary algorithm, MOEA) that can be hybridized with local search approaches. SPEA and NSGA-II are frequently used for this type of hybridization [27, 29], but all GA can be hybridized. A list of these approaches can be found in [30]. In general, MOEA constitutes a good base for applying a local search in population-based MOPs. In hybrid approaches, we still find the traditional genetic operators: selection and crossover. However, the mutation operator is often used to apply the local search.

In the present paper, the genetic algorithm that is used as a base for MARPOCS is NSGA-II [27], for its simplicity and its good performance. In the next section, we will describe the local search that will be used in MARPOCS.

3.4.2 Local search procedure to improve solutions

Dominance-based local search approaches [31] deliver very good results. These LS can be integrated in memetic algorithms. For example, M-PAES is a modified version of PAES, which is adapted to memetic algorithm [20]. However, adding a sophisticated local search metaheuristic to a genetic algorithm can make the memetic algorithm complicated in terms of implementation and execution. Hence, a simple local search algorithm combined with genetic operators can be more efficient in solving MOPs.

There are different approaches for applying a local search algorithm in memetic algorithm. Each objective can sequentially processed separately, or all the objectives can be treated together by random weighting to push the found solutions to Pareto front. When the algorithm processes solutions during local search procedure, it might encounter new non-dominated solutions. These new solutions can be simply copied to the archived set or they can be added to the current population.

To adapt the problem in the present paper to local search procedure, it is turned into a mono-objective problem where the fitness value is equal to the weighted sum of the different criteria with respect to the chosen search direction, which is randomly generated and defined by the weights vector ω .

Each non-dominated solution found during the local search is added to the archive. In that way, the archive is built directly using the local search procedure. The last solution found in the local search procedure (local

optimum) is added to the population used in genetic algorithm in order to be used to build the next generation.

The local search strategy used in this paper belongs to the first improvement hill climbing (FIHC) category. This type of local search successively accepts the first neighbor that has a better fitness. This way, a partial neighborhood exploration is done during each LS iteration. The algorithm stops when it cannot find any new improved solution, which is the local optimum. A local search is formally defined by the couple (Ω, V) where Ω is the set of feasible solutions (decision space) and V is the neighborhood structure $V: \Omega \rightarrow 2^\Omega$ that assigns for each solution $s \in \Omega$ a set of neighbors $V(s)$. Therefore, the algorithm stops when the algorithm finds the local optimum s^* , that is reached if and only if $\forall s \in V(s^*), f(s) \geq f(s^*)$.

In MARPOCS, the neighborhood of a solution s is defined as the set of solutions s' that can be obtained by adding or removing an operation from the solution. In a more formal way, this relation can be expressed as:

$$V(s) = \{s' \mid d(s, s') = 1\}, \quad (9)$$

where $d(s, s')$ is the Hamming distance between s and s' . When all the operations in s and s' are the same then $d(s, s') = 0$. While $d(s, s') = 1$ when there is one operation in difference between s and s' . We also note that changing the time of an operation goes into two steps: removing the operation then adding it in a different time step.

In MARPOCS, the local search procedure is performed through three steps as shown in “Algorithm 4”:

In the first step, the algorithm uses a list of the relocation operations that can decrease the number of rejected demands, obtained using the algorithm described in [19]. The algorithm calculates the weighted fitness of the solution after adding each operation from this list until finding an improvement. The algorithm will keep any operation that improves the fitness of the solution while it disregards the others.

Then in the second step, the local search procedure calculates the weighted fitness after removing each operation of the solution. If the fitness is not improved, the algorithm restores that operation.

In addition, in the third step the algorithm evaluates the fitness of the solution after changing the time of each operation (forward or backward through the time). If the fitness is not improved, the algorithm restores the old value of time.

Every time the fitness is improved, the solution is added to the archive if it is not dominated by any other solutions while all dominated solutions are removed. However, just one solution will be modified in the genetic population.

Algorithm 4 Local search procedure for the proposed memetic algorithm

Require: s the offspring that we want to improve */* s the child that comes after a crossover operation */*

```

1: /* Start Step 1 */
2:  $O \leftarrow \text{getGreedyOperations}(s)$  /* Get the operations that reduce the number of rejected demands using the policy 3 */
3:  $\Omega \leftarrow \text{generateRandomWeights}()$  /*  $\omega$  the vector of random weights of objective functions, such as  $\sum_{i=1}^n \omega_i = 1$  */
4:  $\min \leftarrow \sum_{i=1}^n \omega_i \times f_i(s)$ 
5: for each  $op \in O$  do
6:    $s.\text{add}(op)$  /* Integrate the operation in the solution */
7:    $f_s \leftarrow \sum_{i=1}^n \omega_i \times f_i(s)$  /* Calculate the fitness value after the modifications */
8:   if  $f_s > \min$  then
9:      $s.\text{remove}(op)$  /* Remove the operation if the fitness is not improved */
10:  else
11:     $\min \leftarrow f_s$ 
12:     $\text{addNonDominated}(s)$  /* Add  $s$  to the archive if it is not dominated by any other solution and remove the dominated ones */
13:  end if
14: end for
15: /* Start Step 2 */
16: for each  $op \in s$  do
17:    $s.\text{remove}(op)$  /* Remove the operation from the solution */
18:    $f_s \leftarrow \sum_{i=1}^n \omega_i \times f_i(s)$ 
19:   if  $f_s < \min$  then
20:      $\text{addNonDominated}(s)$ 
21:      $\min \leftarrow f_s$ 
22:   else
23:      $s.\text{add}(op)$ 
24:   end if
25: end for
26: /* Start Step 3 */
27: for each  $op \in s$  do
28:    $op.\text{changeTime}()$  /* Change the time of the operation by incrementing or decrementing it */
29:    $f_s \leftarrow \sum_{i=1}^n \omega_i \times f_i(s)$ 
30:   if  $f_s < \min$  then
31:      $\text{addNonDominated}(s)$ 
32:      $\min \leftarrow f_s$ 
33:   else
34:      $op.\text{undoChangeTime}()$  /* Restore the time change */
35:   end if
36: end for

```

3.4.3 Memetic algorithm process

In a memetic algorithm, there are three main steps:

- Selection of parents p_1 and p_2 .
- Crossover between the two parents that gives a new solution s .
- Local search process based on the randomly generated vector that represents the search direction $\mathbf{v}$ to obtain the local optimum solution s^* . This vector is defined by the weights vector $\omega = (\omega_1, \dots, \omega_k)$. During the local search, the algorithm accepts all solutions s_i obtained through the neighbor structure and that have better fitness based on the aggregation function.

The vector that defines the progress of the local search allows a good exploration of the search space. In fact, the vector $\mathbf{v}$ and the current solution s_i , define a hyperplane orthogonal to $\mathbf{v}$ and passes by s_i . During the local search,

all the solutions s_{i+1} that are above of this hyperplane improve the aggregated functions and are accepted. Figure 3 illustrates this LS for two objectives, in that case the hyperplane is a line.

The algorithm below describes the steps used in MARPOCS:

Algorithm 5 Proposed memetic algorithm

Require: N the size of the population, $nbGen$ the number of generations */* A time limit or any termination condition can be used to replace the $nbGen$ */*

```

1:  $P \leftarrow \text{init}(N)$  /* Initialize the population  $P$  with  $N$  random individuals */
2:  $Q \leftarrow \emptyset$  /* An empty population for children */
3:  $A \leftarrow \emptyset$  /* Initialize an empty archive */
4:  $\text{eval}(P)$  /* Evaluate objectives for each individual */
5: for  $i = 1$  to  $nbGen$  do
6:    $P \leftarrow P \cup Q$ 
7:    $\text{assignRank}(P)$  /* Based on Pareto dominance */
8:   for each front  $f \in P$  do
9:      $\text{setCrowdingDist}(f)$ 
10:  end for
11:   $\text{sort}(P)$  /* By rank and in each rank by the crowding distance */
12:   $P \leftarrow P[0 : N]$ 
13:   $Q \leftarrow \text{buildChildren}(P)$ 
14:   $\text{improveChildren}(Q)$  /* Following the local search FIHC */
15: end for

```

The main difference between this algorithm and NSGA-II, is the integration of the new function *improveChildren*. This function applies the local search on all the individuals of the population Q . The selection and the crossover operators are applied in the function *buildChildren*. However, the mutation operator is omitted since it is replaced by the local search.

4 Experimentation and results

4.1 Mobility data

The mobility data used for this study comes from a real context that consists of survey data and socio-economical information collected by professional for regional planning purposes. It describes people mobility flows in a region of 20 km $\times$ 10 km in Paris. This area is divided into a grid of equal cell sizes. Each cell has two characteristics:

- Terrain type: to describe the dominant structure type of the area associated to the cell (roads, buildings, houses, business center, commercial center, etc.)

- Attraction weight: based on the terrain type and survey data, this information attributes a dynamic attraction weight to each cell for each 15 min of the day

People mobility between different cells is represented by a 3D matrix $F = (f_{i,j,t})$, where $f_{i,j,t}$ represents the number of people who want to move from cell i to cell j at time t . We consider t to be a period of 15 min during the day, which makes 96 time periods. Then the flow mobility data is plotted on a map using GIS shapefiles. As a result, 400 cells have been detected as a potential origin or destination point knowing that some cells are eliminated because of their geographical nature, e.g., lakes, plains, etc. The final flow mobility data consists of $400 \times 400 \times 96$ elements, which makes 15,360,000 records to represent how people move during the day.

In [32], we developed a platform for locating stations for a carsharing system. This platform takes advantage of the mobility data that we described earlier in Sect. 4.1. We used this platform to generate the data for this study. For each generated dataset, we use four parameters:

1. Total number of cars in the system.
2. Total number of stations in the system.
3. Average number of trips per car during a day.
4. Number of parking spaces in each station.

The platform is used to find the optimal locations of stations in a new carsharing system within an urban area. Given a combination of the described parameters, this platform locates the stations in a way that covers the maximum user demands while respecting the predefined objectives. The platform is used to constitute the data that will be used in the optimization process. It is important to mention that the traffic condition is omitted in this study, because the used platform does not provide precise information about it. It is really very important to include this variable when implementing the study, and it can be estimated statistically in case of real system data.

Fig. 3 Selection and crossover followed by a local search

4.2 JMetal framework

JMetal is an open source object oriented framework based on Java for multiobjective optimization using metaheuristics [33]. This framework is provided with many state-of-the-art algorithms, benchmark problems, and different quality indicators that are used to evaluate the performance of MOP. We use this framework to develop, experiment and to study our algorithms in order to solve the multiobjective optimization carsharing problem. The framework also facilitates the development of experimental studies with special tools that are dedicated for this purpose. It also offers the feature of statistical analysis of the experimentation results. In addition, this framework allows running experiments in parallel to exploit the multi-core processors in order to reduce the execution time of the used algorithms.

In our study, we use JMetal 4.5 on a PC with Intel Core i5-3550 CPU (3.30 GHz) and 16 GB of RAM.

4.3 Evaluating the quality of a set of solutions

In mono-objective optimization, the evaluation of algorithms is easy since solutions have only one fitness value. However, this is not the case in MOP where each solution has k values for k objectives, and each algorithm generates trade-off solutions that constitute an approximation of the Pareto Front. Thus to evaluate two algorithms, we have to compare two sets of solutions generated by each algorithm. When all the solutions generated by an algorithm B are dominated by a solution of algorithm A , in this special case we can say that A is better than B . However, it is not easy to compare two set of solutions where some solutions of A dominate some solution of B and vice versa. Therefore, we generally use performance indicators that can be used to evaluate an approximation of Pareto front. These performance indicators evaluate the solutions based on two criteria: convergence and diversity. There are many quality indicators that are proposed in the literature such as hypervolume, generational distance, inverted generational distance, epsilon, spread, etc. Each quality indicator is intended to measure one or both of the criteria at the same time as shown in Table 2. In another perspective, quality indicators can be classified into two main categories: unary indicators that assign a score for a set of solutions, and binary indicators that compare two sets of solutions and assign a score to a set relatively to the other one.

4.3.1 Epsilon

The epsilon indicator has been presented in [34]. This indicator has two versions: multiplicative and additive. In its additive version, the epsilon indicator is used to

calculate the minimum translation factor ε that should be added to every solution of an approximation A in order to dominate solutions of the Pareto front. This indicator can be defined as follows:

$$I_{\varepsilon+}^1(A) = \varepsilon \in \mathbb{R} \{ \forall z^2 \in PF^* \exists z^1 \in A : z^1 \prec_{\varepsilon} z^2 \}, \quad (10)$$

where $z^1 \prec_{\varepsilon} z^2$ if and only if $\forall 1 \leq i \leq n : z_i^1 < \varepsilon + z_i^2$

The smaller the value of ε is, the better the approximation is. It is good to note that a normalization of the objective functions is required in order to obtain efficient results when the scale is not the same for all the objective functions.

4.3.2 Generational distance (GD)

The generational distance indicator was proposed by [35]. This indicator measures the average distance (in objective space) between the approximation set and the Pareto front as shown in the Eq. 11. The smaller the value of GD is, the better the approximation is. When all the approximation solutions are found in the Pareto front the value of GD is 0.

$$GD = \frac{\sqrt{\sum_{i=1}^n d_i^2}}{n}, \quad (11)$$

with n being the number of solutions contained in the approximation set and d_i the Euclidean distance in the objective space between each solution of the approximation set and its nearest solution in the Pareto front.

4.3.3 Inverted generated distance (IGD)

IGD is a modified version of GD where this indicator calculates the average distance between the Pareto front and the approximation set instead of the average distance between the approximation set and the Pareto front.

$$IGD = \frac{\sqrt{\sum_{i=1}^{n'} d_i^2}}{n'}, \quad (12)$$

where n' is the number of solutions contained in the Pareto front and d_i the Euclidean distance in the objective space between each solution of the Pareto front and its nearest solution in the approximation set.

4.3.4 Spread (Δ)

The spread indicator measures the spread level of the solutions that belong to an approximation set according to the Pareto front. It is defined as below:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}}, \quad (13)$$

where d_i is the distance between two adjacent solutions, $\bar{d}$ is the average value of these distances, and d_f and d_l are the Euclidean distances between the extreme solutions of the Pareto front and the approximation set. A spread value of zero refers to an ideal distribution (Fig. 4).

4.3.5 Hypervolume (HV)

The hypervolume indicator computes the volume in the objective space of the space covered by the set of non-dominated solutions A of an approximation set. This indicator is very popular in the literature since it reflects the quality of an approximation based on two criteria: diversity and convergence. In its unary form, the hypervolume is calculated based on a reference point Z_{ref} ("anti-optimal" or nadir point) which refers to the worst possible point in the objective space as shown in Fig. 5 where we deal with a minimization problem. A hypercube v_i is formed of each solution $s_i \in A$ and the reference point Z_{ref} , as diagonal corners. The hypervolume is the union of all hypercubes:

$$HV = \text{volume} \left(\bigcup_{i=1}^{|A|} v_i \right). \quad (14)$$

4.4 Reference front

We use quality indicators to compare the performance of algorithms in MOPs. In most cases, we compare the approximation set with the Pareto front. However, in real-world problems, usually the Pareto front is unknown. To cope with this problem, we build an approximation of the Pareto by combining all the results of all the runs tested for our problem. Thereby we can compare the relative performance of the algorithms. This is our approach for the car relocation problem.

4.5 Algorithms parameters

In the present paper, we compare two algorithms for solving the multiobjective carsharing problem: NSGA-II and memetic. We have to choose the most suitable values for the genetic parameters such as mutation probability, crossover probability and population size. A good choice of these parameters highly affect the performance of the algorithm. For this sake, we define different combinations of these three parameters then we use a class that is defined in the JMetal platform for this purpose. The concept is to run the algorithms many times with different parameters combinations. After that, we compare the results using the values of performance indicators that are defined in the platform and precisely the hypervolume (HV). The configuration that is associated with the best performance

Table 2 Quality indicators with regard to convergence and diversity

Measure	Epsilon	GD	Spread	IGD	HV
Convergence	X	X		X	X
Diversity			X	X	X

indicators values is considered to have the best parameters settings. Since the proposed algorithms are stochastic, we run our algorithms 5 times for each combination before comparing the results. After that, we compare the results of each algorithm for each configuration based on HV.

The values of the parameters that are tested together for NSGA-II and MA are shown in Table 3.

Therefore we test 10 levels of population, 5 levels of crossover and 6 levels of mutation probabilities. That is 300 combinations ($10 \times 5 \times 6$) tested for NSGA-II algorithm and 50 combinations (10×5) for the memetic algorithm, since the local search replaces the mutation for all the individuals and is applied in a systematic way to offspring.

We start by running the NSGA-II algorithm over the 300 combinations with 5 runs for each. The average and the standard variation of the hypervolume are used to rank the performance of these different parameters combinations based on the hypervolume indicator. The best 5 combinations are shown in Table 4.

On the other side, we run the 50 experiments of the memetic algorithm with 5 runs for each combination parameters. Here also, we rank the experiments based of the HV indicator. The best 5 combinations are shown in Table 5.

4.6 Comparison of algorithm results

In order to evaluate the performance of the multiobjective approach for the car relocation problem, we use the best combination of parameters obtained for each algorithm in Sect. 4.5. The results concern a carsharing system that is composed of 18 stations with 10 parking spaces each, having a fleet of 88 cars with an average of 12 trips per car during the whole day. In the same way as for fixing the parameters, each algorithm is run 5 times using the best combination of parameters found in Tables 4 and 5.

We compare the approximation sets obtained using both algorithms based on the best combination chosen for each algorithm in Sect. 4.5. Since it is not easy to compare solutions of multiobjective problems in a 3-D space as shown in Fig. 6, we propose to plot each two objectives on a 2-D chart in order to make the comparison more readable.

In Fig. 7, we see a two objectives comparison between the solutions generated by NSGA-II and those generated by the MARPOCS. The number of rejected demands is plotted

Fig. 4 Distances from extreme solutions [33]

against the number of jockeys. As we can see, most of the solutions obtained with the MA dominate the solutions obtained using NSGA-II algorithm. The initial total number of rejected demand without any relocation operation is 312. As shown, the same number of jockeys may be associated with different numbers of rejected demands. Obviously, this can be explained by the fact that the third objective is not shown. The lowest values of remaining rejected demands for each number of jockeys are associated with the highest values of working time for this number of jockeys. The comparison also shows that MA succeeds to generate solutions that cover the best values for the rejected demands objective. Starting the thirteenth jockey, the algorithm starts to generate solutions with zero rejected demands, which is consistent with the value of the number of jockeys required to solve all the rejected demands in [19] using an exact or a greedy approach.

In Fig. 8, the working time is plotted against the number of jockeys. Here also, the number of jockeys can be associated with different values of working time depending on the value of the third objective. This proves the ability of the algorithm to generate different solutions for the same number of rejected demands depending on the dedicated working time. For example, in Fig. 7, the highest values of remaining rejected demands (> 130 rejected demands for number of jockeys > 21) are associated with the lowest values of total working time shown in Fig. 8 (total working time < 100 for the number of jockeys > 21). This is the interest of a multiobjective approach: Giving the possibility to the operators to choose between several equivalent solutions. Sometimes, some solutions are not interesting. For example, solutions with high number of jockeys, high rejected demands and low total working time, may not be useful for the operator. But those solutions will be listed by

Fig. 5 Hypervolume indicator (HV)

a multiobjective algorithm since they are not dominated on at least one criteria.

It is important to notice that the memetic approach, as shown in Figs. 7 and 8, failed to obtain solutions with low remaining rejected demands when the number of jockeys exceeds 21. Firstly, this is not a problem, since the algorithm was able to find a set of solutions with 0 remaining rejected demand by using a lower number of jockeys. Secondly, this can be explained by the fact that the algorithm did not have the needed time to find those solutions.

In the last combination of objectives, we plot the working time against the number of remaining rejected demands (Fig. 9). As we can observe, the solutions quality of MA is much better than those of the NSGA-II algorithm regarding the total working time objective. The MA can generate solutions that solve all the rejected demands using a lower value of working time. It is also important to note that even when NSGA-II uses a greater working time, it cannot go under 31 rejected demands.

From another point of view, In Fig. 9, there is a negative relationship between the rejected demand and total working time. This can be explained by the fact that if we have rejected demands, we need to do relocation operations in order to avoid them. Each relocation operation needs time to be done. For example, we initially have 312 rejected demands in the system; the total working time needed to keep them as they are is zero, and as much as we do relocation operations to decrease the number of rejected demands, the total working time will increase also. In this figure, we see that MARPOCS performs better than NSGA-II. For keeping the same number of rejected demands, MARPOCS finds always a solution with less total working time than the solution found by NSGA-II.

Table 3 Tested values of the parameters for NSGA-II and MA

Population	Crossover probability	Mutation probability
2000	1	0.6
1500	0.9	0.5
1000	0.8	0.4
500	0.7	0.3
300	0.6	0.2
200		0.1
100		
50		
25		
10		

Table 4 NSGA-II: ranking of best 5 combinations based on HV mean values

Rank	Population	Crossover	Mutation	HV
1	1500	0.9	0.6	0.50691
2	1000	0.9	0.6	0.50631
3	1000	1.0	0.4	0.50608
4	1000	1.0	0.6	0.50480
5	1500	1.0	0.4	0.50423

Table 5 MA: ranking of best 5 combinations based on HV mean values

Rank	Population	Crossover	HV
1	300	0.8	0.59923
2	300	0.9	0.59901
3	300	1.0	0.59870
4	300	0.7	0.59868
5	500	0.8	0.59738

From indicators point of view, Fig. 10 shows a comparison of the evolution of the hypervolume indicator and the number of solutions in the archive for NSGA-II and MA. The hypervolume indicator is obtained by comparing the approximation obtained for each algorithm with the reference set obtained earlier by combining the results of all the runs of our algorithms using different parameters settings.

As we can see, the hypervolume of solutions obtained by the MA is always higher than the hypervolume of solutions obtained by NSGA-II. We can also notice that after the first two seconds only, the hypervolume of MA solutions has the same level of the hypervolume of solutions obtained by the NSGA-II solutions after 30 s of algorithm run. After that, the hypervolume related to the MA starts to stagnate after 40 s when it reaches the 0.6 level. While the hypervolume of the NSGA-II algorithm

Fig. 6 Solutions in 3-D space

Fig. 7 Number of rejected demands according to the number of jockeys

Fig. 8 Total working time according to the number of jockeys

starts to stagnate after the 125 s of execution when it reaches the 0.52 level.

On the other side, we notice that the evolution of the number of solutions in both algorithms has the same trend over the time. It is also clear that sometimes the number of solutions in the archive of the NSGA-II algorithm is greater than the number of solutions in MA. However, these

Fig. 9 Total working time according to the number of rejected demands

Fig. 10 Hypervolume indicator (HV)

solutions have of lower quality as indicated by the hypervolume and as will be shown in the next paragraphs.

In Table 6, we observe a comparison between NSGA-II and MA based on different quality indicators. The comparison shows that the MA performs much better than the NSGA-II algorithm with regard to almost all the quality indicators. The number of solutions in the NSGA-II archive is close to that number in MA archive. However, the quality of solutions in MA is better than those of NSGA-II as shown in the quality indicators values. The spread value in NSGA-II is sometimes greater than its value in MA. However, all other quality indicators show that the quality of the MA solutions is better than those of NSGA-II solutions.

It is also worth noting that we run the algorithms through different carsharing configurations, and they proved that in average they needs between few seconds to one minute to converge in case of medium networks (30 stations). This duration should be acceptable to meet the

Table 6 Results for 5 runs of NSGA-II and memetic algorithms after two minutes and 30 seconds of running time

NSGA-II	NbSol	HV	GD	IGD	Spread	Epsilon
1	1160	0.49632	0.00304	0.00332	0.65236	0.11538
2	1086	0.48773	0.00294	0.00344	0.64520	0.11218
3	1076	0.50601	0.00298	0.00303	0.63196	0.12500
4	1083	0.50165	0.00294	0.00314	0.68354	0.10700
5	1258	0.49610	0.00276	0.00297	0.70454	0.12179
Min	1076	0.48773	0.00276	0.00297	0.63196	0.10700
Max	1258	0.50601	0.00304	0.00344	0.70454	0.12500
Mean	1132.6	0.49756	0.00293	0.00318	0.66352	0.11627
SD	77.96	0.00686	0.00011	0.00019	0.02975	0.00725
Memetic	NbSol	HV	GD	IGD	Spread	Epsilon
1	1024	0.59720	0.00049	0.00143	0.64825	0.02000
2	1023	0.59821	0.00047	0.00138	0.71926	0.02000
3	950	0.59809	0.00048	0.00178	0.59643	0.02000
4	973	0.59913	0.00046	0.00119	0.68447	0.01923
5	999	0.59665	0.00046	0.00196	0.61199	0.02000
Min	950	0.59665	0.00046	0.00119	0.59643	0.01923
Max	1024	0.59913	0.00049	0.00196	0.71926	0.02000
Mean	993.8	0.59786	0.00047	0.00155	0.65208	0.01985
SD	32.18	0.00096	0.00001	0.00031	0.05070	0.00034

requirement of real-time scheduling of a carsharing system, since the algorithm gives the plan for the entire day.

link to the Creative Commons license, and indicate if changes were made.

5 Conclusion

In this paper, a new approach is proposed for addressing the one-way car relocation problem based on a multiobjective problem formulation. The proposed model allows to find best trade-off solutions without making assumptions about the cost of a rejected demand, which is not easy to determine. Three objectives are fixed and modeled for the presented problem. To solve this problem, two algorithms are implemented. The first one is a classic NSGA-II that has been adapted to the problem and used as a reference. The second one is a new memetic approach.

The comparison of the approximation sets obtained by both algorithms shows how much the solutions generated by the MARPOCS are better than the solutions generated by NSGA-II. This observation is proved by the comparison of different quality indicators values that are used to compare the performance of each algorithm.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a

References

1. Dror M, Fortin D, Roucairol C (1998) Redistribution of self-service electric cars: a case of pickup and delivery
2. Barth M, Todd M (1999) Simulation model performance analysis of a multiple station shared vehicle system. *Transp Res Part C Emerg Technol* 7(4):237–259
3. Kek AG, Cheu RL, Chor ML (2006) Relocation simulation model for multiple-station shared-use vehicle systems. *Transp Res Rec J Transp Res Board* 1986(1):81–88
4. Kek AG, Cheu RL, Meng Q, Fung CH (2009) A decision support system for vehicle relocation operations in carsharing systems. *Transp Res Part E Logist Transp Rev* 45(1):149–158
5. Cucu T, Ion-Boussier L, Ducq Y, Boussier J-M (2009) Management of a public transportation service: carsharing service. In: *The 6th international conference on theory and practice in performance measurement and management.*, Dunedin, New Zealand
6. Nair R, Miller-Hooks E (2011) Fleet management for vehicle sharing operations. *Transp Sci* 45(4):524–540
7. Smith S, Pavone M, Schwager M, Frazzoli E, Rus D (2013) Rebalancing the rebalancers: optimally routing vehicles and drivers in mobility-on-demand systems. In: *American control conference (ACC)*, 2013, pp 2362–2367
8. DeMaio P (2009) Bike-sharing: history, impacts, models of provision, and future. *J Public Transp* 12(4):41–56
9. Barth M, Todd M, Xue L (2004) User-based vehicle relocation techniques for multiple-station shared-use vehicle systems. *Transp Res Rec* 1887:137–144

10. Uesugi K, Mukai N, Watanabe T (2007) Optimization of vehicle assignment for car sharing system. In: Knowledge-based intelligent information and engineering systems. Springer, New York, pp 1105–1111
11. Mitchell WJ (2010) Reinventing the automobile: personal urban mobility for the 21st century. MIT press, Cambridge
12. Jorge D, Correia GH, Barnhart C (2014) Comparing optimal relocation operations with simulated relocation policies in one-way carsharing systems
13. Chan ND, Shaheen SA (2012) Ridesharing in north america: past, present, and future. *Transp Rev* 32(1):93–112
14. Correia G, Viegas JM (2011) Carpooling and carpool clubs: clarifying concepts and assessing value enhancement possibilities through a stated preference web survey in lisbon, portugal. *Transp Res Part A Policy Pract* 45(2):81–90
15. Boyacı B, Zografos KG, Geroliminis N (2017) An integrated optimization-simulation framework for vehicle and personnel relocations of electric carsharing systems with reservations. *Transp Res Part B Methodol* 95:214–237
16. Huang K, de Almeida Correia GH, An K (2018) Solving the station-based one-way carsharing network planning problem with relocations and non-linear demand. *Transp Res Part C Emerg Technol* 90:1–17
17. Xu M, Meng Q, Liu Z (2018) Electric vehicle fleet size and trip pricing for one-way carsharing services considering vehicle relocation and personnel assignment. *Transp Res Part B Methodol* 111:60–82
18. Zakaria R, Dib M, Moalic L, Caminada A (2018) Insights on car relocation operations in one-way carsharing systems. *Int J Adv Comput Sci Appl* 9(7):281–290
19. Zakaria R, Dib M, Moalic L, Caminada A (2014) Car relocation for carsharing service: comparison of cplex and greedy search. In: IEEE symposium on computational intelligence in vehicles and transportation systems (CIVTS). IEEE, pp 51–58
20. Knowles JD, Corne DW (2000) M-paes: a memetic algorithm for multiobjective optimization. In: Evolutionary computation, Proceedings of the 2000 Congress on, volume 1. IEEE, pp 325–332
21. Jaszkiewicz A (2002) Genetic local search for multi-objective combinatorial optimization. *Eur J Oper Res* 137(1):50–71
22. Deb K, Goel T (2001) Controlled elitist non-dominated sorting genetic algorithms for better convergence. In: Evolutionary multi-criterion optimization. Springer, New York, pp 67–81
23. Golberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison Wesley, Boston
24. Zitzler E, Künzli S (2004) Indicator-based selection in multiobjective search. In: Parallel problem solving from nature-PPSN VIII. Springer, New York, pp 832–842
25. Fonseca CM, Fleming PJ et al (1993) Genetic algorithms for multiobjective optimization: formulation discussion and generalization. *ICGA* 93:416–423
26. Srinivas N, Deb K (1994) Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolut Comput* 2(3):221–248
27. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolut Comput IEEE Trans* 6(2):182–197
28. Moscato P et al. (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. In: Caltech concurrent computation program, C3P Report, p 826
29. Zitzler E, Laumanns M, Thiele L, Zitzler E, Zitzler E, Thiele L, Thiele L (2001) Spea2: improving the strength pareto evolutionary algorithm
30. Coello CC, Lamont GB, Van Veldhuizen DA (2007) Evolutionary algorithms for solving multi-objective problems. Springer, New York
31. Liefvooghe A, Jourdan L, Talbi E-G (2010) Metaheuristics and cooperative approaches for the bi-objective ring star problem. *Comput Oper Res* 37(6):1033–1044
32. Moalic L, Lamrous S, Caminada A (2013) A multiobjective memetic algorithm for solving the carsharing problem, In: *WORLD COMP'13-The 2013 world congress in computer science, computer engineering, and applied*. vol. 1, pp 877–883
33. Durillo JJ, Nebro AJ (2011) Jmetal: a java framework for multi-objective optimization. *Adv Eng Softw* 42(10):760–771
34. Zitzler E, Thiele L, Laumanns M, Fonseca CM, Da Fonseca VG (2003) Performance assessment of multiobjective optimizers: an analysis and review. *Evolut Comput IEEE Trans* 7(2):117–132
35. Van Veldhuizen DA, Lamont GB (1998) Multiobjective evolutionary algorithm research: a history and analysis. In: Technical report, Citeseer