Computers & Operations Research 71 (2016) 149–162

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A destroy and repair algorithm for the Bike sharing Rebalancing Problem

Mauro Dell'Amico [a], Manuel Iori [a], Stefano Novellani [a,b,*], Thomas Stützle [c]

[a] DISMI, University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy
[b] "Guglielmo Marconi" - DEI, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
[c] IRIDIA, Université Libre de Bruxelles (ULB), 50, Av. F. Roosevelt, CP 194/6 B-1050 Brussels, Belgium

ARTICLE INFO

Available online 22 January 2016

Keywords:
Bike sharing
Rebalancing
Destroy and repair
Speed-up techniques
Branch-and-cut

ABSTRACT

In this paper, we deal with the Bike sharing Rebalancing Problem (BRP), which is the problem of driving a fleet of capacitated vehicles to redistribute bicycles among the stations of a bike sharing system. We tackle the BRP with a destroy and repair metaheuristic algorithm, which makes use of a new effective constructive heuristic and of several local search procedures. The computational effort required for the neighborhood explorations is reduced by means of a set of techniques based on the properties of feasible BRP solutions. In addition, the algorithm is adapted to solve the one-commodity Pickup and Delivery Vehicle Routing Problem with maximum Duration (1-PDVRPD), which is the variant of the BRP in which a maximum duration is imposed on each route.

Extensive computational results on instances from the literature and on newly-collected large-size real-world instances are provided. Our destroy and repair algorithm compares very well with respect to an exact branch-and-cut algorithm and a previous metaheuristic algorithm in the literature. It improves several best-known solutions, providing high quality results on both problem variants.

1. Introduction

Bike sharing systems are public or private systems designed to increase the use of bicycles and decrease congestion, to solve the last mile problem, and to provide a mobility service to the users where other means of transportation are not available. These systems were born in the 1960s in Amsterdam (see, e.g., DeMaio [1]) and spread all over the world now, counting more than 700 operating systems and more than 200 systems planned, under construction, or about to be implemented (see e.g., DeMaio and Meddin [2]).

Bike sharing systems are composed of several bike stations located in different sites around the city, and each station is composed of a number of bike slots where bicycles can be collected or returned. In a balanced situation, each bike station must have a certain number of empty slots, to allow arrivals, and a certain number of full slots, to allow departures.

Let us define the *level of occupation* of a station as the number of bicycles present in that station, and the *balanced level of occupation* as the level of occupation that the system operator desires in a station. After a certain amount of time from the beginning of the service, the users will have moved the bicycles among the stations of the system and the level of occupation will have gone far from the balanced one. For example, the stations on the top of a hill will be typically empty and the stations at the bottom will be typically full. This situation creates inefficiency in the system such that users cannot collect or return bicycles when they need to. The system operators want the stations to be brought back to their balanced levels of occupation to avoid the inefficiency created by full and/or empty stations, so they perform a redistribution of bicycles that is called *rebalancing*. The rebalancing is performed by capacitated vehicles and it is normally required at the end of the day, when the system is closed or when the use of the system can be considered negligible. In this case, the rebalancing is called static. Some bike sharing operators may require that the rebalancing is performed when the system is open, incurring in a situation that is called dynamic rebalancing. In this paper we study the static case.

The *Bike sharing Rebalancing Problem* (BRP) is a problem related to the static rebalancing of a bike sharing system, that requires to drive a fleet of homogeneous and capacitated vehicles to rebalance the stations of a bike sharing system at minimum cost. In this paper, we present a new constructive heuristic and a set of efficient local searches that are included into a destroy and repair

* Corresponding author at: DISMI, University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy.
E-mail addresses: mauro.dellamico@unimore.it (M. Dell'Amico), manuel.iori@unimore.it (M. Iori), stefano.novellani@unimore.it (S. Novellani), stuetzle@ulb.ac.be (T. Stützle).

http://dx.doi.org/10.1016/j.cor.2016.01.011
0305-0548/

metaheuristic algorithm for the BRP. This new algorithm provides shorter solving times for the easiest instances and better upper bounds for the hardest instances when compared to the branch-and-cut algorithm developed by Dell'Amico et al. [3] and good quality solutions for newly collected, large real-world instances.

Moreover, we adapted the metaheuristic algorithm to the *one-commodity Pickup and Delivery Vehicle Routing Problem with maximum Duration* (1-PDVRPD), a variation of the BRP where a maximum duration constraint is imposed to each route. Instances of the 1-PDVRPD from the literature have been solved and the best known solutions have been strongly improved. To solve the 1-PDVRPD we have also adapted the branch-and-cut algorithm of Dell'Amico et al. [3], by efficiently handling the maximum duration limit as a set of constraints of exponential size. This method applies to two-index formulations for general vehicle routing problems, so we believe it is a further interesting contribution in this field of research.

The paper is structured as follows. In Section 2, we formally define the BRP and the 1-PDVRPD. Mathematical models are proposed for both problems and a brief literature review is reported. In Section 3, we describe some properties of the BRP that allow us to speed up local searches. In Section 4, we present the framework of the proposed destroy and repair algorithm for the BRP and we describe its components. In Section 5, we describe the adaptation of the destroy and repair and of the branch-and-cut algorithms to the 1-PDVRPD. In Section 6, extensive computational results on newly collected real instances and on literature instances are presented. Section 7 concludes the work.

2. Problem description

The *Bike sharing Rebalancing Problem* (BRP) is modeled on a complete digraph $G=(V,A)$, where $V=\{0,1,\ldots,n\}$ is the set of vertices including the depot (vertex 0) and the n stations (vertices $1,\ldots,n$), and A is the set of arcs between each pair of vertices. Let c_{ij} be the non-negative cost associated with the arc $(i,j)\in A$. We assume that the triangular inequality valid for the cost matrix. For each vertex $i\in V$ a request q_i is given, with $q_0=0$. Requests can be either positive or negative. If a station has a positive request, we call it a pickup station; if, instead, it has a negative request, we call it a delivery station. The quantities picked up at pickup stations can be used to respond to requests of delivery stations or can be returned directly to the depot. Vehicles can leave the depot not necessarily empty, if needed. The objective is to drive a fleet of m identical vehicles of capacity Q available at the depot to respond to requests and to minimize the total cost of the traversed arcs. As commonly assumed in the related literature (see, e.g., Dell'Amico et al. [3]), we impose that a station with request $q_i=0$ must be visited, even if this implies that no bike has to be dropped off or picked up there. This is imposed to allow the routine inspection of bikes and stations. The case in which stations with null request have to be skipped can be simply obtained by removing them in a preprocessing phase.

2.1. Formulation for the BRP

In this section, we present a *mixed integer linear programming* (MILP) formulation for the BRP. The starting point is *Formulation F3* by Dell'Amico et al. [3] built on the *multiple Traveling Salesman Problem* (m-TSP), in which at most m uncapacitated vehicles based at a central depot have to visit a set of vertices, with the constraint that each vertex is visited exactly once. By defining a binary variable x_{ij}, taking value 1 if arc (i,j) is traveled by a vehicle, and 0 otherwise, the BRP can be modeled as (1)–(6).

$$\min z=\sum_{i\in V}\sum_{j\in V}c_{ij}x_{ij} \tag{1}$$

$$\sum_{i\in V}x_{ij}=1 \quad j\in V\setminus\{0\} \tag{2}$$

$$\sum_{i\in V}x_{ji}=1 \quad j\in V\setminus\{0\} \tag{3}$$

$$\sum_{j\in V}x_{0j}\le m \tag{4}$$

$$\sum_{i\in S}\sum_{j\in S}x_{ij}\le |S|-\max\left\{1,\left\lceil\frac{\left|\sum_{i\in S}q_i\right|}{Q}\right\rceil\right\} \quad S\subseteq V\setminus\{0\}, S\neq\varnothing \tag{5}$$

$$x_{ij}\in\{0,1\} \quad i,j\in V. \tag{6}$$

Objective function (1) minimizes the traveling costs. Constraints (2) and (3) impose that every vertex but the depot is visited once. Constraints (4) ensure that at most m vehicles leave the depot. To guarantee the feasibility of a solution with respect to the BRP, we need to substitute the typical subtour elimination constraints by the family of constraints (5) in the m-TSP formulation, so as to ensure that requests are satisfied and vehicles capacities are not exceeded. Constraints (5) are similar to the *generalized subtour elimination constraints* for the CVRP. They state that, for each subset S of vertices, the number of arcs with both tail and head in S should not exceed the cardinality of S minus the minimum number of vehicles required to serve S. Following Hernández-Pérez and Salazar-González [4], an estimation of the minimum number of vehicles is simply obtained by computing the absolute value of the sum of the requests, dividing it by the vehicle capacity and then rounding up the result. This value can be zero when the sum of the q_i is null; in such a case, the value one is used instead, because at least one vehicle is needed (notice that S does not contain the depot) to impose the connectivity of the solution. Constraints (5) are exponentially many, so the above formulation can be solved in *branch-and-cut* (B&C) fashion, by invoking a separation procedure to divide the violated ones from the non-violated ones, and then adding the violated cuts to the model in an iterative way. For details we refer to Dell'Amico et al. [3] and to Section 5.2 below.

2.2. A generalization of the BRP: the 1-PDVRPD

The *one-commodity Pickup and Delivery Vehicle Routing Problem with maximum Duration* (1-PDVRPD) is a generalization of the BRP in which a maximum duration constraint is imposed for the routes. The problem was solved by Shi et al. [5], in which is imposed a maximum distance D to each route, where the distance is computed as the sum of the traveling costs. In this paper, we solve a more general version of the 1-PDVRPD where the maximum distance D is considered as a maximum duration in time, T, where a time t_{ij} of traveling the arc $(i,j)\in A$ is introduced and it is proportional to the cost between i and j, i.e., $c_{ij}=\tau t_{ij}$, with τ a non-negative parameter. Moreover, we consider a service time s_i in each station that depends on the quantity to be picked up or dropped off, imposing that $s_i=\sigma|q_i|$, with σ a non-negative parameter. We solve the version of the 1-PDVRPD that takes into account the traveling time t_{ij}, the service time s_i, and the maximum duration of a route T because it fits better with the BRP. The 1-PDVRPD proposed by Shi et. al [5] is a particular case of our version, where $\tau=1,\sigma=0$, and $T=D$.

For modeling the 1-PDVRPD as a MILP we need to express the maximum duration constraint. To this aim, we define a path P as an ordered sequence of vertices $P(0),P(1),\ldots,P(|P|)$. We use $\mathcal{R}$ to identify the set of routes, i.e., the set of paths P starting and ending at the depot,

$(P(0)=P(|P|)=0)$. We guarantee that each route $P \in \mathcal{R}$ does not exceed the maximum duration limit, by considering the traveling times on the arcs and the service times on the visited stations, as follows:

$$\sum_{i=0}^{|P|-1} (t_{P(i)P(i+1)} + s_{P(i+1)}) x_{P(i)P(i+1)} \leq T \quad P \in \mathcal{R}. \tag{7}$$

Note that constraints (7) are exponentially many, and, hence, as already observed for (5), we need a separation procedure that divides violated constraints from non-violated ones. This is discussed in Section 5.2 below.

2.3. Prior work

The BRP and the 1-PDVRPD belong to the class of *Pickup and Delivery Vehicle Routing Problems* (PDVRP), and generalize the *Capacitated Vehicle Routing Problem* (CVRP). Thus, they are both strongly NP-hard problems.

According to the classification introduced by Berbeglia et al. [6] and followed by Battarra et al. [7] the BRP is a *Many-to-Many* (M-M) vehicle routing problem, where the origins and destinations of the requests are multiple. The simplest M-M PDVRP is the *one-commodity Pickup and Delivery Traveling Salesman Problem* (1-PDTSP), which was introduced by Hernández-Pérez and Salazar-González [8]. The 1-PDTSP aims at traveling a single capacitated vehicle to meet the multiple requests of a single commodity and minimize the costs. Hernández-Pérez and Salazar-González [4] presented asymmetric and symmetric formulations for the 1-PDTSP and solved the symmetric one by means of a B&C algorithm (which was later improved by Hernández-Pérez and Salazar-González [9]). Hernández-Pérez et al. [10,11] presented two simple heuristics and a variable neighborhood descent. Other metaheuristic algorithms for the 1-PDTSP were proposed, lately, by Martinovic et al. [12] (iterated modified simulated annealing), Zhao et al. [13] (genetic algorithm), Hosny and Mumford [14] (hybrid variable neighborhood search and simulated annealing approach), and Mladenović et al. [15] (variable neighborhood search).

The BRP is a generalization of the 1-PDTSP to the case of multiple vehicles, and was formally introduced by Dell'Amico et al. [3]. They proposed four formulations for the BRP and solved them by B&C. They evaluated the algorithms on 65 real-world instances and 15 random instances solving to optimality all instances with up to 51 vertices.

The multiple vehicle case with a maximum duration limit imposed for every route was formally introduced by Shi et al. [5]. They proposed a three-index formulation for which they did not report any computational evaluation. They also presented a genetic algorithm and tested it on a set of randomly generated symmetric instances derived from those proposed by Hernández-Pérez and Salazar-González [10].

Many other papers deal with the balancing of bike sharing systems considering different aspects. We report a relevant collection of these papers by dividing them into static rebalancing with split deliveries, rich static rebalancing, and dynamic rebalancing. The *static rebalancing with split deliveries*, i.e., the case where customers can be visited more than once and their requests split, is examined in Benchimol et al. [16]. They presented complexity results, lower bounding techniques, and approximation algorithms for the single vehicle case where the sum of all requests is equal to zero. A computational evaluation of the techniques was not given. Chemla et al. [17] also considered the single vehicle case where split deliveries are allowed, but they imposed a limit on the maximum number of times a vertex can be visited. Stations with null requests can be used as buffers. The authors presented a formulation and a tabu search algorithm, and performed a large series of computational tests. Erdoğan et al. [18] also solved the single vehicle problem with split deliveries by allowing the use of the stations as storages. The authors solved the problem by means of a B&C algorithm, proposed a constructive heuristic, and presented good results on different sets of instances.

We refer to *rich static rebalancing* problems as those problems where the static rebalancing aspect is considered in conjunction with many additional features, such as transshipment, different objectives, multiple depots, inventory policies, and possibly also split deliveries. Raviv et al. [19] defined the *static bicycle repositioning problem* as the problem of minimizing the traveling costs of a fleet of heterogeneous vehicles and the users' dissatisfaction that is linked to the inventory level of each station. They presented two MILP formulations allowing, respectively, limited and unlimited split delivery and transshipment, and considering also a maximum duration for each route. They also developed exact and heuristic methods to solve the presented formulations. Papazek et al. [20] solved another rich static problem defined as the *balancing bicycle sharing system problem*, which intends to find the routes of an heterogeneous fleet of vehicles to minimize the deviation from the target level of request of each station. They also considered, as secondary objectives, the duration of the routes and the number of loading actions performed. The problem allows split deliveries and transshipment and it is imposed that the vehicles return empty to the depot. The authors proposed a greedy constructive heuristic and some metaheuristic algorithms. Di Gaspero et al. [21] solved a similar rich static rebalancing problem where split deliveries are allowed but intermediate storages are not. The problem considers multiple depots and multiple vehicles, and a maximum duration time imposed on each vehicle. The aim is to find routes that operate pickups and deliveries to bring the level of occupation in each station as close as possible to the balanced one and that minimize the total traveling time. The authors introduced two different constraint programming models and proposed two branching strategies. Then they included the constraint programming models in a large neighborhood search approach obtaining good computational results. Schuijbroek et al. [22] combined the inventory policy and the routing optimization. The authors solved the problem of minimizing the makespan of the routes respecting a lower and a upper bound on the number of bikes required in each station. The stations that do not need a visit can be used as a buffer, the routes can be open (arbitrary routes start and end points) and a limited number of transshipments is allowed. The authors proposed a MILP formulation, a constraint programming approach, and a cluster-first route-second heuristic.

A solution method for the *dynamic rebalancing*, where requests can change during the rebalancing operations, was presented by Contardo et al. [23]. The objective is to maximize the serviced requests by using a set of vehicles. The authors presented a formulation using discretized time and solved it heuristically with Dantzig–Wolfe and Benders decompositions.

3. Properties of feasible paths for the BRP

In this section, we present some properties of feasible BRP paths that allow us to speed up the computation of several procedures that are at the basis of our metaheuristic algorithm. These procedures are formally defined in Section 4 and they are all based on the following set of operators:

- *Remove* a vertex i from its current position in the solution.
- *Insert* a vertex i in a position.
- *Move* a vertex i from its position to another one (composition of remove and insert).
- *Swap* two vertices i and j by moving i to the position of j and j to the position of i (once again, composition of remove and insert).
- *Merge* two (partial) routes.

Our speed-up techniques build on the basic route feasibility property presented in [10], and that we resume here briefly. We recall that a path P is an ordered sequence of vertices $P(0), P(1), \ldots, P(|P|)$, while a route is a path that starts and ends at the depot, i.e., $P(0) = P(|P|) = 0$. Let us first compute the cumulative request along a route by using the following recursion

$$l_{P(i)}(P) = \sum_{k=0}^{i} q_{P(k)} \quad \text{for } i = 0, 1, \ldots, |P|. \tag{8}$$

In other words, $l_{P(i)}(P)$ gives the value of the load on the vehicle *after* visiting vertex $P(i)$ of route P, if the vehicle left the depot with no load. Negative $l_{P(i)}(P)$ values may still lead to feasible routes, because the initial load of the vehicle is not restrained to be 0. In particular, a route is feasible if the following inequality is satisfied (see [10]):

$$\max_{i=0}^{|P|-1} \{l_{P(i)}(P)\} - \min_{i=0}^{|P|-1} \{l_{P(i)}(P)\} \le Q. \tag{9}$$

In the following we make use of a parameter Δ_P, that we call the *amount of feasibility* of P and set as

$$\Delta_P = Q - \max_{i=0}^{|P|-1} \{l_{P(i)}(P)\} + \min_{i=0}^{|P|-1} \{l_{P(i)}(P)\}. \tag{10}$$

Intuitively, routes with a small Δ_P value have a tight constraint on the choice of the initial load. This parameter can be used to define some sufficient conditions, called Δ-*checks* in the following, for quick feasibility checks of the previously introduced neighborhood operators.

Property 1. *If $i \notin P$ and $|q_i| \le \Delta_P$, then vertex i can be feasibly inserted in any position of route P.*

Proof. Let P' be the route obtained after the insertion of vertex i in route P. By applying (9), P' is feasible if

$$\max_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} - \min_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} \le Q. \tag{11}$$

To prove that (11) holds if the hypothesis is satisfied, let us consider two cases:

- $q_i \ge 0$. In this case, with respect to the route P, the insertion of vertex i may increase or keep unchanged both the max and the min terms in (11). The maximum difference between the two emerges when the max increases by the maximum possible quantity, i.e., by q_i, and the min does not change. We can thus bound this difference as

$$\begin{aligned}
&\max_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} - \min_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} \le \left(\max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} + q_i \right) \\
&\quad - \min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} = \max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} + q_i - \min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} \\
&\quad \le \max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} + \Delta_P - \min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} = Q,
\end{aligned}$$

  where the last two steps follows, respectively, from the hypothesis and from (10).
- $q_i < 0$. In this case we consider instead that the maximum difference between the two terms in (11) is obtained when the min term decreases by q_i and the max remains unchanged, thus we can state that

$$\begin{aligned}
&\max_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} - \min_{k=0}^{|P'|-1} \{l_{P'(k)}(P')\} \le \max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} \\
&\quad - \left(\min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} + q_i \right) = \max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} - \min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} - q_i \\
&\quad \le \max_{k=0}^{|P|-1} \{l_{P(k)}(P)\} - \min_{k=0}^{|P|-1} \{l_{P(k)}(P)\} + \Delta_P = Q,
\end{aligned}$$

and the thesis is verified.□

Similar proofs, reported in Appendix A, lead to the following results.

Property 2. *Given a feasible route P:*

(a) *If $i \in P$ and $|q_i| \le \Delta_P$, then vertex i can be feasibly removed from route P.*
(b) *If $i \in P$ and $|q_i| \le \Delta_P$, then vertex i can be feasibly moved in any position along route P.*
(c) *If $i, j \in P$ and $|q_i - q_j| \le \Delta_P$, then the swap of vertices i and j is feasible.*

Moreover, given a pair of feasible routes P and R:

(d) *If $i \in P, j \in R$, and $|q_i - q_j| \le \min\{\Delta_P, \Delta_R\}$, then the swap of vertices i and j is feasible.*

Proof. See Appendix A.□

When the quick Δ-checks are not enough to certify feasibility of a move, an exact check must be performed. We consider more elaborated data structures to reduce the computational complexity of the exact check. In this sense, our contribution follows a well established line of research on difficult combinatorial problems, arising in areas such as vehicle routing with time windows and scheduling with release and due dates (see, e.g., Ergun and Orlin [24], Liao et al. [25], and Ibaraki et al. [26]).

We use the concept of *load windows* that express the feasible intervals of load that can be carried on the vehicle before or after visiting a vertex along a given route, by extending the basic result in [10]. In particular, we use

$$\begin{aligned}
F_{P(i)}(P) = \left[\underline{f}_{P(i)}(P), \overline{f}_{P(i)}(P) \right] = \Bigg[& l_{P(i)}(P) - \min_{j=0}^{i} \{l_{P(j)}(P)\}, l_{P(i)}(P) + Q \\
& - \max_{j=0}^{i} \{l_{P(j)}(P)\} \Bigg]
\end{aligned} \tag{12}$$

to define the *forward load window* along route P, i.e., the feasible interval for the load on the vehicle after leaving vertex $P(i)$, and supposing that $P(i)$ is the last vertex of the route before returning to the depot. The computation of (12) can be performed in linear time by using a simple structure that keeps track of the minimum and maximum values without recalculating them from scratch. Note that the amount of feasibility of P can be quickly computed as $\Delta_P = \overline{f}_{P(|P|-1)}(P) - \underline{f}_{P(|P|-1)}(P)$.

We similarly define the *backward load window* along route P, as $B_{P(i)}(P) = \left[\underline{b}_{P(i)}(P), \overline{b}_{P(i)}(P) \right]$. This load window gives the feasible interval for the load on the vehicle *before* it visits vertex $P(i)$, and considering that $P(i)$ is the first vertex in the route after the depot. This can be computed by first using a recursion that gives the cumulative load $r_{P(i)}$ in the reverse route starting from the depot and visiting the vertices in the order $P(|P|), P(|P|-1), \ldots, P(i)$. Formally

$$r_{P(i)}(P) = -\sum_{k=|P|}^{i} q_{P(k)} \quad \text{for } i = |P|, |P|-1, \ldots, 0. \tag{13}$$

It is worth recalling that for a route P ending at the depot $P(|P|) = 0$. We thus compute, again in linear time, the backward load window as

$$\begin{aligned}
B_{P(i)}(P) = \left[\underline{b}_{P(i)}(P), \overline{b}_{P(i)}(P) \right] = \Bigg[& r_{P(i)}(P) - \min_{j=i}^{|P|} \{r_{P(j)}(P)\}, r_{P(i)}(P) \\
& + Q - \max_{j=i}^{|P|} \{r_{P(j)}(P)\} \Bigg].
\end{aligned} \tag{14}$$

The forward and backward windows are used to construct the following property.

Property 3. *Two disjoint and feasible routes P and R can be merged in a feasible route $P \oplus R$, where the last vertex of P before returning to*

the depot, $P(|P|-1)$, is followed by the first vertex of R after the depot, $R(1)$, if and only if

$F_{P(|P|-1)}(P) \cap B_{R(1)}(R) \neq \emptyset$.

Proof. See Appendix A.

By applying similar reasonings, we obtain the following additional results.

Property 4. *Given two feasible routes P and R:*

a) *Removing the first vertex after the depot or the last vertex before the depot from P is always feasible.*
b) *Removing a vertex $P(i)$ from P is feasible if and only if $F_{P(i-1)}(P) \cap B_{P(i+1)}(P) \neq \emptyset$.*
c) *Let I be the single-vertex route formed by vertex i, i.e., $(0,i,0)$. The insertion of i into P after vertex $P(p)$ is feasible if and only if the two following conditions are satisfied: (1) i can be feasibly inserted after the first part of P, i.e., $F_{P(p)}(P) \cap B_i(I) \neq \emptyset$; (2) $F_i(P) \cap B_{P(p+1)}(P) \neq \emptyset$, where $F_i(P) = \left[\max\left\{\underline{f}_{P(p)}(P), \underline{b}_i(I)\right\} + l_i(I), \min\left\{\overline{f}_{P(p)}(P), \overline{b}_i(I)\right\} + l_i(I)\right]$. For sake of clarity, we underline that $B_i(I) = [0, Q-q_i]$, if $q_i \geq 0$, and that $B_i(I) = [-q_i, Q]$, if $q_i < 0$.*
d) *Let $I = (0,i,0)$ and $J = (0,j,0)$, and suppose $i \in P$ (let $i = P(h)$) and $j \in R$ (let $j = R(k)$). The swap of i with j is feasible if and only if the following conditions are satisfied for the two routes: (1) $F_{P(h-1)}(P) \cap B_j(J) \neq \emptyset$; (2) $F_j(P) \cap B_{P(h+1)}(P) \neq \emptyset$, where $F_j(P) = \left[\max\left\{\underline{f}_{P(h-1)}(P), \underline{b}_j(J)\right\} + l_j(J), \min\left\{\overline{f}_{P(h-1)}(P), \overline{b}_j(J)\right\} + l_j(J)\right]$; (3) $F_{R(k-1)}(R) \cap B_i(I) \neq \emptyset$; (4) $F_i(R) \cap B_{R(k+1)}(R) \neq \emptyset$, where $F_i(R) = \left[\max\left\{\underline{f}_{R(k-1)}(R), \underline{b}_i(I)\right\} + l_i(I), \min\left\{\overline{f}_{R(k-1)}(R), \overline{b}_i(I)\right\} + l_i(I)\right]$.*

We conclude this section by discussing the update of the load windows. Let us suppose that a merging of two routes P and R has been performed, in this order, and consider the intersection between the forward load window of $P(|P|-1)$, and the backward load window of $R(1)$, that is: $F_{P(|P|-1)}(P) \cap B_{R(1)}(R) = \left[\max\left\{\underline{f}_{P(|P|-1)}(P), \underline{b}_{R(1)}(R)\right\}, \min\left\{\overline{f}_{P(|P|-1)}(P), \overline{b}_{R(1)}(R)\right\}\right] = \left[\underline{\phi}, \overline{\phi}\right]$. It is convenient to update the existing forward and backward load windows, without recomputing them from scratch. To update the forward load window for the vertices of route R we can compute: $F_{R(i)}(P \oplus R) = \left[\underline{\phi} + l_{R(i)}(R), \overline{\phi} + l_{R(i)}(R)\right], i = 1, \ldots, |R|-1$, while for the vertices of route P the forward load window does not change. To update the backward load windows for the vertices of route P we can compute: $B_{P(j)}(P \oplus R) = \left[\underline{\phi} + r_{P(j)}(P), \overline{\phi} + r_{P(j)}(P)\right], j = 1, \ldots, |P|-1$, while for the vertices of route R the backward load window does not change.

4. Destroy and repair algorithm

The overall framework of the algorithm that we use to solve the BRP, called DR_BRP, is reported in Algorithm 1. It mainly consists of an iterated destroy and repair mechanism, enriched with local search procedures. The principle of using solution destruction with a subsequent solution re-construction or repair has been popularized in recent years by Ropke and Pisinger [27], in the context of large neighborhood search. This principle has been proposed, even in more sophisticated ways with respect to our work, in a number of different algorithms using names such as simulated annealing (see, e.g., Jacobs and Brusco [28]), ruin-and-recreate (see, e.g., Schrimpf et al. [29]), large neighborhood search (see, e.g., Shaw [30]), iterated greedy (see, e.g., Ruiz and Stützle [31]), iterative constructive search (see, e.g., Richmond and Beasley [32]), or iterative flattening (see, e.g., Cesta et al. [33]). We first construct a feasible initial solution by using a greedy algorithm (*Savings&Losses*, described in Section 4.1) and we refine it by using a set of local search procedures (explained in Section 4.4). This solution is then iteratively destroyed and repaired by using in order *DestroyProcedure* and *RepairProcedure* (presented in Sections 4.2 and 4.3, respectively), and again improved by the local search procedures, until a stopping criterion is reached. Note that our algorithm works with routes that are feasible with respect to the capacity constraint, but accepts solutions in which the number of routes is larger than m. This number is implicitly minimized by several of the adopted local search procedures.

Algorithm 1. DR_BRP.

Algorithm DR_BRP
Savings&Losses $\rightarrow (x^*, z^*)$
Local searches $(x^*) \rightarrow (x_{ls}, z_{ls})$
if $z_{ls} < z^*$ **then**
  $z^* \leftarrow z_{ls}$ and $x^* \leftarrow x_{ls}$
end if
repeat
  DestroyProcedure $(x_{ls}) \rightarrow (\tilde{x})$
  RepairProcedure $(\tilde{x}) \rightarrow (x')$
  Local searches $(x') \rightarrow (x_{ls}, z_{ls})$
  if $z_{ls} < z^*$ **then**
    $z^* \leftarrow z_{ls}$ and $x^* \leftarrow x_{ls}$
  end if
until Stopping criterion
return x^* and z^*

4.1. Constructive algorithm

The constructive algorithm that we developed starts from the well-known *Savings* algorithm by Clarke and Wright [34], but adapts it to the new problem at hand by introducing the concept of *loss of flexibility*. We start by building n routes, each containing a single vertex. We then iteratively select two routes and merge them into a single one, until no more merging is possible.

To select the pair of routes to be merged, we take into account all possible combinations. When considering the merging of routes, say, P and R, we evaluate the feasibility of the resulting route $P \oplus R$ by making use of Property 3. If feasible, then we evaluate the saving obtained in the cost function, if any, as $S_{P \oplus R} = c_{0,R(1)} + c_{P(|P|-1),0} - c_{P(|P|-1),R(1)}$. We also evaluate the loss of flexibility induced by the merging, as the difference between the amount of feasibility (as defined in Section 3) of the resulting route, $\Delta_{P \oplus R}$, and that of the two original routes, Δ_P and Δ_R, computed as $L_{P \oplus R} = -(\Delta_P + \Delta_R - 2\Delta_{P \oplus R})$. In practice, the more negative is the value of $L_{P \oplus R}$ the more the size of the resulting load window would be consistently reduced with respect to the sizes of the original windows, and so the resulting route would be harder to be feasibly merged with other routes in the successive iterations. We use an evaluation function that takes both terms into account, as

$$E_{P \oplus R} = \alpha S_{P \oplus R} + (1-\alpha) L_{P \oplus R}, \tag{15}$$

where α is a parameter of the algorithm that takes values between 0 and 1. The feasible merging of two routes leading to the highest value according to (15) is then selected. The value taken by α in our computational experiments is discussed in Section 6.

We can note that only the backward load window $P(1)$ and the forward load window of $R(|R|-1)$ are needed to determine the feasibility of merging the two routes. These can be computed efficiently inside Savings&Losses as follows. Let us call $F_{P(|P|-1)}(P) \cap B_{R(1)}(R) = \left[\underline{\phi}, \overline{\phi}\right]$, then we can compute: $F_{P \oplus R(|P \oplus R|-1)}(P \oplus R) =$

$\left[\underline{\phi}+l_{R(|R|-1)}(R), \overline{\phi}+l_{R(|R|-1)}(R)\right]$ and $B_{P\oplus R(1)}(P\oplus R)=\left[\underline{\phi}+r_{P(1)}(P), \overline{\phi}+r_{P(1)}(P)\right]$. Moreover, $r_{P(1)}(P)=-l_{R(|R|-1)}(R)$ and $l_{P\oplus R}(|P\oplus R|-1)(P\oplus R)=l_{R(|R|-1)}(R)+l_{P(|P|-1)}(P)$.

4.2. DestroyProcedure

To insert diversification into the metaheuristic algorithm, we make use of a component called *DestroyProcedure*. Similarly to *Random Removal*, presented by Ropke and Pisinger [27], DestroyProcedure randomly selects a number of vertices, one at a time, independently one from the other and with uniform probability, and removes them from the current solution. The number of vertices to be removed is also selected randomly with a uniform probability in an interval defined by two parameters π and δ; more precisely: $[\max\{3, \pi-\delta\pi\}, \pi+\delta\pi]$. The values taken by the parameters π and δ were set after computational tests, as described in Section 6. Each selected vertex is removed from its current route by applying the remove operator introduced in Section 3. If the selected vertex is the first one (right after the depot) or the last one (right before the depot) of a route, its removal can be operated without causing infeasibility on the remaining route because of Property 4-(a). In all other cases, we check the feasibility of the remaining route by checking Properties 2-(a) and 4-(b). If the remaining route is feasible, we perform the removal of the selected vertex; if it is infeasible, we perform the removal and divide the remaining route into the two separate routes defined by the removal (which are then feasible again because of Property 4-(b)).

4.3. RepairProcedure

After using the DestroyProcedure, we are left with a partial solution where not every vertex is allocated to a route. To restore feasibility, we apply two repair procedures that use the insert operator to include the non-assigned vertices into existing or newly formed routes. In details:

1. *RepairInsertion*: We evaluate the feasibility and the cost of inserting each one of the non-assigned vertices in any position in any route, plus the option of creating a new route containing only the considered vertex. Among the feasible options, we select the one having minimum cost, and then re-iterate with the next non-assigned vertex until all of them are assigned. To check the feasibility of the insertion we make use of Properties 1 and 4-(c) of Section 3.
2. *Savings&Losses*: We use the constructive algorithm of Section 4.1, by considering the existing routes in the partial solution and each non-assigned vertex as if it were a single-vertex route. The merging of the routes follows the previously described process.

According to preliminary computational tests, we set RepairProcedure to execute at each iteration of the algorithm either RepairInsertion or Savings&Losses, by alternating them.

4.4. Local search procedures

To improve the solution obtained by the constructive algorithm and/or after a destroy and repair phase, we implemented several local search procedures based on the operators introduced in Section 3. Apart from the Δ-checks, the load windows, and the fast update techniques previously described, we also make use of a simple but effective idea called *Don't look bit*, that we describe here for the insert operator. If we determine that the insertion of a vertex into a route is infeasible, then we keep track, in a matrix, of this infeasibility as long as the route is not modified by other procedures, so as to avoid useless feasibility re-evaluations. A slight variation applies to the swap neighborhood. We now give the details of the local search procedures that we implemented:

1. *Move*: Select a vertex, remove it from its current position and insert it into another position, either in the same route or in a different one, maintaining the sequence. Feasibility is quickly checked by means of Properties 1, 2-(a), 2-(b), 4-(b), and 4-(c). If the selected vertex is the only vertex in its route, then the route is deleted.
2. *Or-opt*(κ): Select at most κ consecutive vertices in a route, remove them from their current position and insert them into another position, either in the same route or in a different one. Let us call P the route from which we remove the κ vertices and R the one in which we insert them. The feasibility of the removal is checked by merging the two remaining subroutes of P as discussed in Property 3. The feasibility of the insertion is instead performed in $O(\kappa)$, by first evaluating the κ forward and backward load windows of the removed vertices as if they were a single route, and then merging them with the first part of R by using again Property 3. If this is feasible, we update the load windows and check the feasibility of merging this new subroute with the second part of R. The value κ is a parameter of the algorithm to be defined after computational tests, as described in Section 6.
3. *Swap*(1,1): Select two vertices in the same route or in different routes and swap them. The feasibility is checked by applying Properties 2-(c), 2-(d), and 4-(d).
4. *Swap*(2,2): It extends the previous local search by selecting two pairs of consecutive vertices and swapping them, maintaining the same order of the vertices in each pair. The process for checking feasibility is similar to the one described for *Or-opt*(κ), but simpler because only pairs of vertices are moved in this case and no longer lists of κ vertices.
5. *Swap*(1,1,1): Select three vertices, say, v_1, v_2, and v_3, belonging to either, one, two, or three routes, and swap them in the two possible ways (v_1 with v_2, v_2 with v_3, v_3 with v_1, or v_1 with v_3, v_3 with v_2, v_2 with v_1). Use Properties 2-(c), 2-(d), and 4-(d) for feasibility check.
6. *Cross*: Select two routes, P and R, and two positions along the routes. Cut each route right after the selected position, and merge the first part of P with the second part of R, and the first part of R with the second part of P. Use Property 3 for feasibility check. Note that a particular case arises when a position is at the very beginning of its route and the other position is at the very end of the other route, when Cross consequently attempts a merging of the two routes into a single one.
7. *Cross*(3): It extends the previous local search, by selecting three routes, and attempting the two possible ways of cutting and merging them.

The number of routes in the solution is possibly decreased by the procedures *Move*, *Or-opt*(κ), *Cross*, and *Cross*(3). Every procedure is run in best improvement fashion, which is computationally preferable to first improvement on our instances, in this sense we defined and computationally tested the parameter ι, that can take value 1 for the best improvement and 0 for the first improvement. The procedures are inserted in a *Variable Neighborhood Descent* (VND) type framework (see, e.g., Hansen and Mladenović [35]), where they are invoked one after the other according to a given order. Our implementation is a VND type framework because we do not go back to the previous local search procedure if an improvement is found, but we continue invoking local searches following the given order until the last one, then we reiterate from the first local search and follow the given order until no improvement in the current solution is provided by all local search

procedures. This method is similar to the piped VND (see, e.g., den Besten and Stützle [36]), where the procedure stops at the end of the last local search procedure. According to the preliminary computational tests, the order of the local searches was set to 1, 3, 5, 4, 6, 2, and 7.

In terms of complexity, we first note that a straightforward implementation of Move would require $O(n^3)$ time, because the number of possible moves is $O(n^2)$ (considering each vertex to be removed and each insertion position), and evaluating each move would require $O(n)$ for computing feasibility of the at most two routes involved by the move. With the use of the load windows we reduce this complexity to $O(n^2)$, because feasibility is checked in $O(1)$. Similar reasonings apply to the next local searches, leading to the following complexities: $O(n^2)$ for *Or-opt*(κ) considering a fixed value of κ; $O(n^2)$ for *Swap*(1,1) and *Swap*(2,2); $O(n^3)$ for *Swap*(1,1,1); $O(n^2)$ for *Cross*; and $O(n^3)$ for *Cross*(3).

5. Adaptation to the 1-PDVRPD

In this section, we describe how we adapted the algorithm presented in the previous section and the B&C by Dell'Amico et al. [3], both originally developed for the BRP, to include the maximum route duration constraint so as to solve the 1-PDVRPD.

5.1. Destroy and repair for the 1-PDVRPD

To solve the 1-PDVRPD, we must perform additional tests when checking the feasibility of a solution, as the feasibility for the BRP is a necessary but not sufficient condition for the feasibility of the 1-PDVRPD. Thus, after checking that a solution is feasible for the BRP, we evaluate that the sum of the travel and service times of each route does not exceed the maximum duration.

The inclusion of this additional check can be performed without increasing the complexity when using the data structures as above. This inclusion is easy for every component of our algorithm so we do not describe it in details. We mention here the fact that the remove operator is always feasible because of the triangular inequality that we assumed to be valid for the cost matrix (and thus the time matrix) whilst the insert operator needs an additional check on the times of the routes. For local searches based on the merge operator, it is convenient to keep track of the cumulative time (service and travel) from the depot to any vertex in a route. This is useful, for example, when performing a move of *Cross* that attempts to merge the first part of a route P with the second part of a route R. The total duration of the resulting route can be obtained by summing the total time of the first part of P, the time to travel along the connecting arc, and the total time of the second part of R (obtained by subtracting the total time of the first part of R from the total time of R).

5.2. Branch-and-cut for the 1-PDVRPD

To solve to optimality model (1)–(7) for the 1-PDVRPD, a B&C algorithm is required because of the exponential number of constraints. The B&C algorithm that we used is directly derived from the one presented in Dell'Amico et al. [3]. We developed some new valid inequalities for the different features of the problem and several separation procedures have been added to the mentioned B&C algorithm. For the implementation we used the B&C framework of CPLEX 12.2, that solves at every node of an enumeration tree the linear relaxation of an ILP model, and then invokes user-developed separation procedures to possibly add cuts. We adopted strong branching as branching rule.

We first present some valid inequalities that we use to strengthen constraints (7), so as to improve the convergence of the algorithm to the optimum, and then we discuss the separation procedures that we implemented.

5.2.1. Valid inequalities

For the first family of inequalities we need some further notation. Let P be a path. We recall that $P(i)$ denotes the index of the i-th vertex of path P, for $i = 0, 1, \ldots, |P|$, with $P(0) = 0$. A path is called infeasible with respect to the maximum duration if $\sum_{i=0}^{|P|-1}(t_{P(i)P(i+1)} + s_{P(i+1)}) > T$. As in our case the triangular inequality holds for both the cost and time matrix, we strengthen this feasibility check by adding to its left hand side the value $t_{P(|P|),0}$, which gives a lower bound on the time needed to return to the depot.

Let $\mathcal{P}$ be the family of all infeasible paths, the following *infeasible path constraints* are thus valid inequalities for the BRP:

$$\sum_{i=0}^{|P|-1} x_{P(i),P(i+1)} \le |P| - 2 \quad P \in \mathcal{P}. \tag{16}$$

Inequality (16) simply states that, if path P is infeasible then not all the arcs connecting two consecutive vertices of P may belong to a solution. A way to enforce it is to consider the related *tournament constraints*, by taking into account also the arcs connecting non-consecutive vertices of P. Indeed, any arc $(P(i), P(j))$ with $j \neq i+1$ is incompatible with arcs $(P(i), P(i+1))$ and $(P(j-1), P(j))$, because of, respectively, out-degree and in-degree constraints. Hence, we can add the corresponding variable, $x_{P(i),P(j)}$ to the left-hand side of (16), without affecting the right-hand side value. This process can be repeated for all arcs connecting two non-consecutive vertices in the path, with the exception of those arcs leaving the depot, because up to m of them may belong to a feasible solution. We thus obtain the following:

$$x_{0,P(1)} + \sum_{i=1}^{|P|-1} \sum_{j=i+1}^{|P|} x_{P(i),P(j)} \le |P| - 2 \quad P \in \mathcal{P}. \tag{17}$$

Inequalities (16) and (17) enforce the maximum route duration constraints, but are computationally very weak. To strengthen them in some favored cases, we take advantage of a new type of constraint, similar to the well-known capacity cut constraint, which we call *time-packing constraint*.

To express the time-packing constraint family of inequalities we need to introduce a lower bound on the time needed to reach a vertex and serve it. Let us define $lb_i(V) = \min_{j \in V : |q_j + q_i| \le Q} \{t_{ji}\} + s_i, i \in V$. Thus we can add to the model the following inequality (18), which computes a bound on the number of vehicles needed to serve the customers in S while respecting the maximum time constraint. Note that $lb_0(S) = \min_{j \in S}\{t_{j0}\}$ is the lower bound on the time needed to return to the depot. If the summation of all the lower bounds is greater than the maximum time T, then more than one route is needed to serve the subset S.

$$\sum_{i \in S} \sum_{j \in V \setminus S} x_{ij} \ge \left\lceil \frac{\sum_{i \in S} lb_i(V) + lb_0(S)}{T} \right\rceil \quad S \subseteq V \setminus \{0\}, S \neq \emptyset, S \cap \{0\} = \emptyset. \tag{18}$$

5.2.2. Separation procedures

The aim of this subsection is to present the procedures that we use to determine if the valid inequalities that we proposed are violated by a given possibly fractional solution $\overline{x}$.

To separate constraints (18), we first build a supporting graph $\overline{G} = (\overline{V}, \overline{A})$, where the set of vertices is $\overline{V} = V \cup \{n+1\}, \{n+1\}$ is a dummy node, and the set of arcs is $\overline{A} = A' \cup A''$. We have $A' = (i,j) \in A : \overline{x}_{ij} > 0$, and a capacity $\overline{x}_{ij}$ is assigned to every arc $(i,j) \in A'$; moreover $A'' = (n+1, i) : i \in V \setminus \{0\}$. With each arc $(n+1, i) \in A''$ is associated a capacity equal to $lb_i(V)/T$. We then compute

the max flow on $\overline{G}$, using $n+1$ as a source and 0 as a sink. The constraint (18) corresponding to the set S induced by the min-cut is then checked, and, if violated, it is added to the model.

The tournament constraints (17) are separated exactly by generating all possible paths starting from the depot and using a depth-first strategy. We initialize path P with $P(0)=0$, then select the outgoing arc having the largest value of $\overline{x}$, and extend the path to include the head of the selected arc. Every time we add a vertex to the path, we check if it is infeasible. If so, then we add the cut, otherwise we continue extending the path. The path extension continues as long as the sum of the involved $\overline{x}_{ij}$ is large enough to possibly lead to a violated cut, i.e., as long as it is strictly greater than the current cardinality of P subtracted by 2. When this condition is not satisfied anymore, then we backtrack to the previous vertex. Anytime we backtrack, we continue the depth-first search by selecting the next arc with positive value of $\overline{x}$, if any, and then extend the path consequently. In our implementation, whenever we find a violated cut of type (17) for a certain path P, then we also add the corresponding constraint (7), and then we terminate the separation procedure.

The separation procedures are invoked at every node of the enumeration tree in the order in which we described them, to separate (18), (17), and (7). On the basis of computational evidence on the instances that we tested, we stop the separation process as soon as we find a violated cut, if any.

6. Computational results

In this section, we present the computational results for the metaheuristic and the B&C algorithms that we implemented to solve the BRP and the 1-PDVRPD. To evaluate the quality of the metaheuristic algorithm when solving the BRP, we used the instances by Dell'Amico et al. [3] and, replicating their method, we collected some new larger instances that are available online [37]. The new instances use real data of bike sharing systems of the following cities: Brisbane, Milano, Lille, Toulouse, Sevilla, Valencia, Bruxelles, Lyon, Barcelona, and London. Some characteristics of the new instances are reported in Table 1, where we depict the name of the city, the country in which the bike sharing system is located, and the number of vertices of each instance. We then present the minimal, average, and maximum value, and the standard deviation of the requests and of the transportation costs. To evaluate the performance of our algorithms on the 1-PDVRPD we used the instances proposed by Shi et al. [5].

According to the number of vertices $|V|$, we divided both the BRP and the 1-PDVRPD instances into three sets: *small-size* instances for $|V| \le 50$, *medium-size* instances for $50 < |V| < 100$, and *large-size* instances for $|V| \ge 100$.

We determined the parameter setting by tuning them with the *irace package* [38], which automatically configures optimization algorithms by finding the most appropriate settings on a given set of instances. We used the default irace Package setting using a maximum number of 1000 algorithms runs during the tuning. The stopping criterion for the algorithm runs is set to 10, 600, and 1800 CPU seconds for small-size, medium-size, and large-size instances, respectively. We decided to run the irace Package on a modified set of instances, derived by the three small-size Dublin instances, all the 21 medium-size instances, and the three large-size Minneapolis instances. We generated the modified instances by perturbing the distances c_{ij} choosing new values randomly in $[0.9\,c_{ij}, 1.1\,c_{ij}]$, for all $(i,j) \in A$ (made sure that the triangular inequality is respected), and by perturbing the request q_i of 10% of the vertices choosing the values randomly in $[\max\{q_i-2,-Q\}, \min\{q_i+2,Q\}], i \in V\backslash\{0\}$. Both BRP and 1-PDVRPD instances have been tested with the obtained set of parameters, which is presented in Table 2. In the same table one can find the parameters used for testing the instances. We recall that α and δ can take real values in the interval $[0,1]$, π can take an integer value in the interval $[3,15]$, ι can take value 0 for first improvement and 1 for best improvement, and κ can take an integer value in the interval $[3,|V|]$. All the tests reported in the following have been performed on a Intel Core i3-2100 with 3.10 GHZ by using randomly generated seed values.

6.1. Small-size instances

To evaluate the DR_BRP algorithm on the set of small-size instances we ran it for ten seconds taking note of the time when it found the best solution. Results are reported in Table 3, where the instances are depicted by the name of city, the number of vertices, and the vehicle capacity. The table also reports the value of the optimal solution (z_{opt}) obtained by the B&C presented in Dell'Amico et al. [3] and the time needed to the B&C to run to completion (t). The last five columns refer to algorithm DR_BRP and report the average solution value (z_{avg}) obtained by running the algorithm ten times for a ten seconds time limit, the percentage gap between the optimal and the average solution value ($\%gap_{avg} = 100 \cdot (z_{avg} - z_{opt})/z_{opt}$), the minimal solution value among the ten trials (z_{min}), the percentage gap between the minimal solution value and the optimal one ($\%gap_{min} = 100 \cdot$

Table 2
Parameters used for computational tests.

Parameter	Value	Meaning and use	Section
α	0.7335	Balances the evaluation function components	4.1
π	4	Used to define the interval for the number of vertices to destroy	4.2
δ	0.6582	Used to define the interval for the number of vertices to destroy	4.2
ι	1	Specifies if local searches use best improvement or first improvement	4.4
κ	35	Maximum limit for consecutive vertices to be selected in *Or-opt* (κ)	4.4

Table 1
New BRP instances.

City	Country	$\|V\|$	$\min\{q_i\}$	$\text{avg}\{q_i\}$	$\max\{q_i\}$	$\text{dev}\{q_i\}$	$\min\{c_{ij}\}$	$\text{avg}\{c_{ij}\}$	$\max\{c_{ij}\}$	$\text{dev}\{c_{ij}\}$
Brisbane	Australia	150	−15	1.27	17	5.91	2	3769.07	13,959	2129.98
Milano	Italy	184	−18	0.88	17	8.40	6	3313.63	8126	1422.46
Lille	France	200	−20	−0.42	16	6.72	163	7450.82	18,877	4655.73
Toulouse	France	240	−13	−1.49	12	6.16	6	3943.35	12,459	1904.32
Sevilla	Spain	258	−20	−1.56	10	6.89	2	4652.58	13,510	2305.15
Valencia	Spain	276	−20	−1.50	14	7.64	134	4241.34	13,413	1892.71
Bruxelles	Belgium	304	−13	−0.18	16	8.24	211	5844.93	19,032	2726.70
Lyon	France	336	−20	−0.70	17	7.85	18	4817.44	47,657	3571.38
Barcelona	Spain	410	−17	−2.56	19	9.84	2	4699.91	13,671	2229.90
London	U.K.	564	−28	−0.58	29	9.91	2	5833.37	19,412	3100.69

Table 3
Results on small-size instances.

Instance			B&C		DR_BRP (10 s)				
City	$\|V\|$	Q	z_{opt}	t	z_{avg}	$\%gap_{avg}$	z_{min}	$\%gap_{min}$	t_b
Bari	13	30	14,600	0.02	14,600.0	0.00	14,600	0.00	0.00
Bari	13	20	15,700	0.02	15,700.0	0.00	15,700	0.00	0.00
Bari	13	10	20,600	0.03	20,600.0	0.00	20,600	0.00	0.00
Reggio Emilia	14	30	16,900	0.02	16,900.0	0.00	16,900	0.00	0.00
Reggio Emilia	14	20	23,200	0.03	23,200.0	0.00	23,200	0.00	0.00
Reggio Emilia	14	10	32,500	0.05	32,500.0	0.00	32,500	0.00	0.01
Bergamo	15	30	12,600	0.03	12,600.0	0.00	12,600	0.00	0.01
Bergamo	15	20	12,700	0.00	12,700.0	0.00	12,700	0.00	0.01
Bergamo	15	12	13,500	0.11	13,500.0	0.00	13,500	0.00	0.00
Parma	15	30	29,000	0.02	29,000.0	0.00	29,000	0.00	0.00
Parma	15	20	29,000	0.02	29,000.0	0.00	29,000	0.00	0.00
Parma	15	10	32,500	0.05	32,500.0	0.00	32,500	0.00	0.00
Treviso	18	30	29,259	0.05	29,259.0	0.00	29,259	0.00	0.04
Treviso	18	20	29,259	0.03	29,259.0	0.00	29,259	0.00	0.04
Treviso	18	10	31,443	0.09	31,443.0	0.00	31,443	0.00	0.08
La Spezia	20	30	20,746	0.03	20,746.0	0.00	20,746	0.00	0.03
La Spezia	20	20	20,746	0.03	20,746.0	0.00	20,746	0.00	0.03
La Spezia	20	10	22,811	0.09	22,811.0	0.00	22,811	0.00	0.12
Buenos Aires	21	30	76,999	0.37	76,999.0	0.00	76,999	0.00	0.03
Buenos Aires	21	20	91,619	3.58	91,619.2	0.00	91,619	0.00	4.20
Ottawa	21	30	16,202	0.02	16,202.0	0.00	16,202	0.00	0.00
Ottawa	21	20	16,202	0.02	16,202.0	0.00	16,202	0.00	0.00
Ottawa	21	10	17,576	0.11	17,576.0	0.00	17,576	0.00	0.02
San Antonio	23	30	22,982	0.08	22,982.0	0.00	22,982	0.00	0.00
San Antonio	23	20	24,007	0.09	24,007.0	0.00	24,007	0.00	0.05
San Antonio	23	10	40,149	1.06	40,149.0	0.00	40,149	0.00	0.37
Brescia	27	30	30,300	0.06	30,300.0	0.00	30,300	0.00	0.07
Brescia	27	20	31,100	0.20	31,100.0	0.00	31,100	0.00	0.09
Brescia	27	11	35,200	1.37	35,200.0	0.00	35,200	0.00	0.39
Roma	28	30	61,900	0.84	61,900.0	0.00	61,900	0.00	0.36
Roma	28	20	66,600	1.72	66,670.0	0.11	66,600	0.00	3.46
Roma	28	18	68,300	0.58	68,300.0	0.00	68,300	0.00	0.00
Madison	28	30	29,246	0.02	29,246.0	0.00	29,246	0.00	0.04
Madison	28	20	29,839	0.05	29,839.0	0.00	29,839	0.00	0.04
Madison	28	10	33,848	0.53	33,848.0	0.00	33,848	0.00	0.20
Guadalajara	41	30	57,476	0.22	57,476.0	0.00	57,476	0.00	2.14
Guadalajara	41	20	59,493	0.34	59,493.0	0.00	59,493	0.00	1.48
Guadalajara	41	11	64,981	1.79	64,981.0	0.00	64,981	0.00	2.41
Dublin	45	30	33,548	6.05	33,595.4	0.14	33,548	0.00	2.29
Dublin	45	20	39,786	76.72	39,817.2	0.08	39,786	0.00	3.73
Dublin	45	11	54,392	610.80	55,000.6	1.12	54,392	0.00	5.28
Avg.				17.25		0.04		0.00	0.66

$(z_{min} - z_{opt})/z_{opt}$), and the average time needed to obtain the best solution by the metaheuristic algorithm (t_b).

One can notice that DR_BRP finds the optimal solution in all ten trials for 37 out of 41 instances in times that are competitive with the ones needed by the B&C algorithm, and obtains the optimal solution at least once for the remaining four instances of the set. The average gap of 0.04% from the optimal solution within an average time of 0.66 s is extremely promising.

6.2. Medium-size instances

The computational results for the medium-size instances of the BRP are reported in Table 4. Some information on the B&C algorithm by Dell'Amico et al. [3] is reported, such as the best lower and upper bounds, the percentage gap between them ($gap = 100 \cdot (UB - LB)/UB$), and the time needed to get these values in a time limit set to one hour. Then we present z_{avg}, the average solution obtained by running the algorithm ten times for ten minutes, and the percentage gaps between z_{avg} and the lower and upper bounds from the B&C, $\%gap_{LB} = 100 \cdot (z_{avg} - LB)/LB$ and $\%gap_{UB} = 100 \cdot (z_{avg} - UB)/UB$, respectively. In Table 4 are also presented the minimal values z_{min} obtained by the algorithm on the ten trials and the gaps between z_{min} and the bounds of the B&C computed in the same form as for z_{avg}. We also show the average time needed to obtain the best solution with a time limit set to 600 s in t_b.

One can notice that DR_BRP can procure the optimal solution for six instances out of eight for which an optimal solution was known. It is worth noticing the improvement on the upper bound obtained for the instances Ciudad de Mexico (90,30) and Ciudad de Mexico (90,20), where DR_BRP improves the B&C solution on average by 17.49% and 18.72%, respectively, and in the best case by 18.08% and 18.98%, respectively.

We can see that on average the metaheuristic algorithm finds solutions 2.68% above the lower bound obtained by the B&C algorithm and improves the upper bound by 2.81%. The best solution cost is only 2.28% above the lower bound and improves the upper bound by 3.17%.

6.3. Large-size instances

In Table 5, we report the results for the large-size BRP instances. The tests have been performed similarly to those for the medium-size instances making use of the same parameter values, but with a time limit of 1800 s. The columns depicted in Table 5

Table 4
Results on medium-size instances.

Instance			B&C				DR_BRP (10 min)						
							Avg				*Min*		
City	$\|V\|$	Q	*LB*	*UB*	%*gap*	*t*	z_{avg}	$\%gap_{LB}$	$\%gap_{UB}$	*t_b*	z_{min}	$\%gap_{LB}$	$\%gap_{UB}$
Denver	51	30	51,583	51,583	0.00	0.67	51,583.0	0.00	0.00	0.48	51,583.0	0.00	0.00
Denver	51	20	53,465	53,465	0.00	25.33	53,465.0	0.00	0.00	24.68	53,465.0	0.00	0.00
Denver	51	10	67,459	67,459	0.00	231.52	67,459.0	0.00	0.00	102.99	67,459.0	0.00	0.00
Rio de J.	55	30	122,547	122,547	0.00	65.57	122,582.1	0.03	0.03	198.66	122,547.0	0.00	0.00
Rio de J.	55	20	155,446	156,140	0.44	3600.00	155,992.7	0.35	−0.09	304.13	155,517.0	0.05	−0.40
Rio de J.	55	10	253,690	259,049	2.07	3600.00	257,412.5	1.47	−0.63	241.12	257,147.0	1.36	−0.73
Boston	59	30	65,669	65,669	0.00	28.14	65,669.0	0.00	0.00	16.16	65,669.0	0.00	0.00
Boston	59	20	71,879	71,879	0.00	473.84	72,057.2	0.25	0.25	178.16	71,916.0	0.05	0.05
Boston	59	16	74,790	75,065	0.37	3600.00	75,318.8	0.71	0.34	280.50	75,085.0	0.39	0.03
Torino	75	30	47,634	47,634	0.00	13.79	47,634.0	0.00	0.00	246.44	47,634.0	0.00	0.00
Torino	75	20	50,204	50,204	0.00	859.69	51,026.0	1.64	1.64	251.83	50,438.0	0.47	0.47
Torino	75	10	58,814	64,797	9.23	3600.00	62,031.6	5.47	−4.27	303.85	61,717.0	4.94	−4.75
Toronto	80	30	40,794	41,549	1.82	3600.00	41,783.5	2.43	0.56	201.49	41,390.0	1.46	−0.38
Toronto	80	20	42,621	47,898	11.02	3600.00	46,876.6	9.98	−2.13	269.37	46,631.0	9.41	−2.65
Toronto	80	12	54,238	60,763	10.74	3600.00	58,878.7	8.56	−3.10	321.39	58,539.0	7.93	−3.66
Miami	82	30	152,229	156,104	2.48	3600.00	154,344.7	1.39	−1.13	335.05	154,038.0	1.19	−1.32
Miami	82	20	209,379	229,237	8.66	3600.00	215,167.1	2.76	−6.14	265.13	214,250.0	2.33	−6.54
Miami	82	10	390,536	415,762	6.07	3600.00	402,746.8	3.13	−3.13	300.28	397,921.0	1.89	−4.29
C. de Mex.	90	30	67,894	88,227	23.05	3600.00	72,797.5	7.22	−17.49	213.91	72,279.0	6.46	−18.08
C. de Mex.	90	20	88,952	116,418	23.59	3600.00	94,621.6	6.37	−18.72	254.22	94,319.0	6.03	−18.98
C. de Mex.	90	17	99,714	109,573	9.00	3600.00	104,213.7	4.51	−4.89	359.59	103,658.0	3.96	−5.40
Avg.					5.17	2309.45		2.68	−2.81	222.35		2.28	−3.17

Table 5
Results on large-size instances.

Instance			B&C (60 min)			DR_BRP (30 min)						
						Avg				*Min*		
City	$\|V\|$	Q	*LB*	*UB*	%*gap*	z_{avg}	$\%gap_{LB}$	$\%gap_{UB}$	*t_b*	z_{min}	$\%gap_{LB}$	$\%gap_{UB}$
Minneapolis	116	30	136,148	137,843	1.23	139,874.3	2.74	1.47	745.28	138,467.0	1.70	0.45
Minneapolis	116	20	157,736	186,449	15.40	166,797.0	5.74	−10.54	1003.72	166,150.0	5.33	−10.89
Minneapolis	116	10	246,133	298,886	17.65	264,335.2	7.40	−11.56	946.37	262,936.0	6.83	−12.03
Brisbane	150	30	108,275	158,043	31.49	115,949.2	7.09	−26.63	742.81	115,120.0	6.32	−27.16
Brisbane	150	20	132,419	196,739	32.69	146,930.0	10.96	−25.32	957.83	146,313.0	10.49	−25.63
Brisbane	150	17	147,236	234,210	37.14	160,385.6	8.93	−31.52	966.44	160,015.0	8.68	−31.68
Milano	184	30	145,245	227,983	36.29	168,931.2	16.31	−25.90	871.18	167,493.0	15.32	−26.53
Milano	184	20	187,175	295,994	36.76	219,558.6	17.30	−25.82	823.76	218,249.0	16.60	−26.27
Milano	184	18	203,716	299,630	32.01	236,394.9	16.04	−21.10	1048.32	234,423.0	15.07	−21.76
Lille	200	30	164,149	231,244	29.01	178,133.5	8.52	−22.97	666.67	176,976.0	7.81	−23.47
Lille	200	20	191,630	440,350	56.48	215,007.8	12.20	−51.17	280.09	213,090.0	11.20	−51.61
Toulouse	240	30	166,653	404,792	58.83	190,146.8	14.10	−53.03	1147.86	188,995.0	13.41	−53.31
Toulouse	240	20	190,739	427,959	55.43	231,062.0	21.14	−46.01	1398.52	228,674.0	19.89	−46.57
Toulouse	240	13	256,036	461,125	44.48	308,982.7	20.68	−32.99	868.98	307,874.0	20.25	−33.23
Sevilla	258	30	194,805	461,011	57.74	227,911.7	16.99	−50.56	898.40	225,076.0	15.54	−51.18
Sevilla	258	20	240,210	516,734	53.51	281,492.2	17.19	−45.52	1054.31	279,990.0	16.56	−45.82
Valencia	276	30	245,979	673,479	63.48	292,262.6	18.82	−56.60	789.54	287,854.0	17.02	−57.26
Valencia	276	20	302,368	698,436	56.71	370,057.4	22.39	−47.02	1064.40	367,201.0	21.44	−47.43
Bruxelles	304	30	255,259	502,920	49.24	315,487.7	23.60	−37.27	851.97	311,097.0	21.88	−38.14
Bruxelles	304	20	301,100	580,594	48.14	379,141.4	25.92	−34.70	835.66	376,387.0	25.00	−35.17
Bruxelles	304	16	348,303	626,721	44.42	426,992.4	22.59	−31.87	1038.33	424,432.0	21.86	−32.28
Lyon	336	30	300,950	580,437	48.15	364,083.4	20.98	−37.27	879.61	360,009.0	19.62	−37.98
Lyon	336	20	356,787	668,971	46.67	437,588.1	22.65	−34.59	1136.03	433,959.0	21.63	−35.13
Barcelona	410	30	311,774	983,627	68.30	545,633.1	75.01	−44.53	458.27	543,929.0	74.46	−44.70
Barcelona	410	20	449,060	1,088,850	58.76	774,818.4	72.54	−28.84	1538.57	771,507.0	71.80	−29.14
Barcelona	410	19	429,327	1,089,070	60.58	805,513.2	87.62	−26.04	1411.27	800,622.0	86.48	−26.49
London	564	30	385,748	1,304,850	70.44	705,232.0	82.82	−45.95	1572.97	699,571.0	81.35	−46.39
London	564	29	363,299	1,339,890	72.89	725,468.0	99.69	−45.86	1447.33	718,026.0	97.64	−46.41
Avg.					45.85		27.78	−33.92	980.16		26.83	−34.40

are the same used in Table 4, but we disregard the column reporting the times of the B&C algorithm because it always reached the time limit of one hour.

One can notice that the B&C algorithm is not giving good bounds for the large-size instances. On the other hand we can comment that in most of the cases DR_BRP can get heuristic

Table 6
Evaluation of local search components.

City	$\|V\|$	Q	$\%gap_1$	$\%gap_2$	$\%gap_3$	$\%gap_4$	$\%gap_5$	$\%gap_6$	$\%gap_7$	$\%gap_{all}$
Rio de Janeiro	55	30	0.03	−0.36	−0.01	−0.02	−0.02	0.00	−0.01	−18.52
Rio de Janeiro	55	20	0.00	−0.15	0.02	−0.04	0.00	−0.10	0.00	−14.91
Rio de Janeiro	55	10	−0.03	−0.17	0.03	−0.07	−0.06	−0.04	−0.01	−9.00
Boston	59	30	0.00	−0.07	0.00	0.00	0.00	0.00	0.00	−18.44
Boston	59	20	−0.03	−0.07	−0.03	0.03	0.02	−0.03	0.00	−20.57
Boston	59	16	−0.08	−1.79	−0.04	−0.36	−0.20	−0.03	−0.13	−34.54
Torino	75	30	0.00	−0.42	−0.02	0.00	0.00	−0.01	−0.01	−16.91
Torino	75	20	−0.66	−1.01	−0.21	−0.13	0.24	−0.20	0.09	−21.04
Torino	75	10	−0.23	−0.59	−0.13	−0.09	0.09	−0.44	0.04	−15.94
Toronto	80	30	0.15	−1.32	0.07	−0.08	−0.14	−0.33	0.05	−34.11
Toronto	80	20	0.49	−1.89	0.39	0.17	0.41	−0.12	0.05	−34.98
Toronto	80	12	−0.33	−1.92	−0.50	−0.40	−0.15	−1.00	−0.43	−26.58
Miami	82	30	0.05	−0.09	0.00	0.04	−0.06	−0.14	0.05	−2.57
Miami	82	20	−0.11	−0.28	−0.17	−0.18	−0.40	−0.22	−0.18	−7.56
Miami	82	10	0.07	0.28	0.18	−0.09	−0.30	0.34	0.26	−10.62
Ciudad de Mexico	90	30	0.12	−1.59	−0.09	0.09	0.10	−0.32	0.06	−23.27
Ciudad de Mexico	90	20	0.06	−0.83	0.07	−0.05	−0.03	−0.24	−0.07	−13.01
Ciudad de Mexico	90	17	−0.20	−1.11	−0.16	−0.09	−0.09	−0.33	−0.01	−20.19
Minneapolis	116	30	−0.13	−1.46	−0.06	0.03	0.14	−0.31	0.23	−17.24
Minneapolis	116	20	−0.40	−0.73	−0.10	−0.13	0.01	−0.35	−0.13	−11.63
Minneapolis	116	10	0.14	−0.71	−0.12	0.13	0.02	−0.30	0.10	−9.40
Brisbane	150	30	0.34	−1.11	0.29	0.19	0.07	0.04	0.03	−11.52
Brisbane	150	20	0.20	−0.68	−0.14	−0.07	−0.12	−0.04	−0.09	−9.04
Brisbane	150	17	0.02	−0.68	−0.15	−0.10	−0.24	−0.06	−0.19	−6.95
Avg.			−0.02	−0.78	−0.04	−0.05	−0.03	−0.18	−0.01	−17.02

Table 7
Results on small-size instances for the 1-PDVRPD.

Instance			B&C	SZG		DR_BRP						
Name	$\|V\|$	Q	z_{opt}	z_{SZG}	t	z_{avg}	z_{min}	$\%gap_{avg}$	$\%gap_{min}$	$\%gap^{SZG}_{avg}$	$\%gap^{SZG}_{min}$	t_b
n20q10A	20	10	5001	5515.4	0.65	5001.0	5001.0	0.00	0.00	−9.33	−9.33	0.10
n30q10A	30	10	6503	6906.0	0.92	6512.8	6503.0	0.15	0.00	−5.69	−5.84	1.91
n40q10A	40	10	7407	7476.4	1.14	7474.8	7407.0	0.92	0.00	−0.02	−0.93	5.60
n50q10A	50	10	7283	9263.5	1.52	7301.9	7283.0	0.26	0.00	−21.18	−21.38	4.43
Avg.					1.06			0.33	0.00	−9.05	−9.37	3.01

solutions whose value is closer to the lower bound than to the upper bound (which is improved by a 33.92%, on average).

6.4. Local search procedures evaluation

In this subsection we furnish with an insight on the contribution of the local search procedures with respect to the complete DR_BRP algorithm. We produced eight versions of our algorithm, by removing one of the seven local search procedures at a time and, then, all of them. Each version was run ten times on the medium-size instances, but disregarding the Denver instances because they are too easy and thus not interesting for this study, and on some large-size instances. In Table 6 we report for each evaluated instance the percentage gap, $\%gap_{(x)}$, between the average solution value obtained by the complete DR_BRP algorithm and the average solution value reached without the local search procedure (x). The index of the local search procedure in Table 6 is the one already used in Section 4.4 (e.g., $\%gap_1$ gives the percentage gap between DR_BRP and DR_BRP without *Move*). In the last column we report the average gap for the DR_BRP without any of the local search procedures. By looking at the average values, reported in the last line of the table, one can notice that *Or-opt*(κ) (index 2) and *Cross* (index 6) have a clear impact on the overall algorithm leading to an improvement in almost each of the considered instances, and that their removal worsens the average solution value by 0.78% and 0.18%, respectively. For the other operators this is less clear, but the negative overall value gives an indication that these operators can be useful. In conclusion, one can see that the importance of the local search procedures is crucial with respect to the complete algorithm. This importance is easy to detect when evaluating the average gap between the solution value of DR_BRP and DR_BRP without local search operators ($\%gap_{all}$), that indicates a 17.02% deterioration of the solution value.

6.5. 1-PDVRPD instances

The DR_BRP algorithm developed for the BRP has been slightly modified as discussed in Section 5.1 to include the maximum duration constraint typical of the 1-PDVRPD, and has been tested to solve the 1-PDVRPD instances available in the literature (taken from Shi et al. [5]).

We reproduced the experiments according to what we did for the BRP instances. In Table 7 we present the results that we obtained on the small-size instances, i.e., those having at most 50 vertices. We ran DR_BRP ten times for ten seconds and set the parameters to the same values used to solve the BRP instances. The metaheuristic by Shi et al. [5] (SZG in the following) was executed on a slower PC running at 1.60 GHz CPU. In Table 7, we report the name of the instance, the number of vertices, and the vehicle capacity. We then include the value of the optimal solution (z_{opt}), obtained thanks to the newly

Table 8
Results on medium-size and large-size instances for the 1-PDVRPD.

Instance			SZG		DR_BRP				
Name	$\|V\|$	Q	z_{SZG}	t	z_{avg}	z_{min}	$\%gap_{avg}^{SZG}$	$\%gap_{min}^{SZG}$	t_b
n60q10A	60	10	9931.6	2.01	8941.1	8939.0	−9.97	−9.99	140.87
n100q10A	100	10	14,379.3	14.85	12,476.9	12,317.0	−13.23	−14.34	855.75
n200q10A	200	10	23,331.7	47.00	19,619.5	19,341.0	−15.91	−17.10	1181.46
n300q10A	300	10	29,805.3	100.25	25,092.3	24,763.0	−15.81	−16.92	808.87
n400q10A	400	10	34,574.4	156.47	34,209.0	33.951.0	−1.06	−1.80	1237.71
n500q10A	500	10	39,872.5	240.06	33,706.5	33,108.0	−15.46	−16.97	1203.76
Avg.				93.44			−11.91	−12.85	904.74

Table 9
Comparison with the SZG algorithm with proportionate time limits.

Name	$\|V\|$	Q	z_{avg}	t_b	z_{SZG}	$\%gap_{avg}^{SZG}$	z_{min}	$\%gap_{min}$
n20q10A	20	10	5001.0	0.10	5515.4	−10.29	5001.0	0.00
n30q10A	30	10	6547.1	0.20	6906.0	−5.48	6503.0	0.67
n40q10A	40	10	7619.3	0.32	7476.4	1.88	7407.0	2.79
n50q10A	50	10	7478.9	0.54	9263.5	−23.86	7283.0	2.62
n60q10A	60	10	9445.1	0.58	9931.6	−5.15	8939.0	5.36
n100q10A	100	10	12,946.6	0.37	14,379.3	−11.07	12,317.0	4.86
n200q10A	200	10	20,423.4	12.89	23,331.7	−14.24	19,341.0	5.30
n300q10A	300	10	26,069.2	43.60	29,805.3	−14.33	24,763.0	5.01
n400q10A	400	10	35,588.0	49.53	34,574.4	2.85	33,764.0	5.13
n500q10A	500	10	34,553.6	121.306	39,872.5	−15.39	32,890.0	4.81
Avg.				22.94		−9.51		3.66

developed B&C algorithm of Section 5.2. In the columns dedicated to the SZG algorithm, we report the best average solution value (z_{SZG}) it produced and the computational time it required to run to completion (t). In the columns for the DR_BRP we show the average and the minimum values of the ten trials of our metaheuristic algorithm (z_{avg} and z_{min}, respectively), their gaps with respect to the optimal solution value ($\%gap_{avg}$ and $\%gap_{min}$), their gaps with respect to z_{SZG} (computed as $\%gap_{avg}^{SZG} = 100 \cdot (z_{avg} - z_{SZG})/z_{SZG}$ and $\%gap_{min}^{SZG} = 100 \cdot (z_{min} - z_{SZG})/z_{SZG}$), and the average time needed to obtain the best solution for each trial (t_b). One can see that with our metaheuristic algorithm we can solve all the small-size instances to optimality for at least one trial (and for all trials on instance n20q10A), within an average time of about 3 s. Moreover, we notice that DR_BRP improves the solutions found by SZG by more than 9% on average, with a reasonable computational effort.

In Table 8, we depict the results of our algorithm on all the medium-size and large-size instances. The DR_BRP algorithm run ten times on each instance by using the same parameters adopted for the BRP instances. It was allowed 10 min on Instance n60q10A (the only medium-size instance according to our classification) and 30 min on the other instances. The B&C was not run because too computationally expensive. Table 8 presents the same columns of Table 7, with the exception of those referring to z_{opt} and to the related gaps. One can notice that our algorithm is able to improve every solution found by SZG, obtaining in around 15 min an average improvement of about 12%.

Finally, we solved the 1-PDVRPD instances by setting as time limits the same times required by the SZG algorithm to run to completion. The PC on which we performed our tests is faster compared to the one used by Shi et al. [5] (3.10 GHz vs 1.60 GHz). Hence, to perform a fair set of tests we multiplied the SZG times by 1.60/3.10 and used the resulting values as termination conditions for DR_BRP. In details, in Table 9 we show the average solution obtained on the ten trials within the time limit computed as previously explained (z_{avg}), and the average time needed to find the best solution (t_b). Then we recall the best average solution of Shi et al. [5] (z_{SZG}) and the gap between z_{avg} and z_{SZG} ($\%gap_{avg}^{SZG} = 100 \cdot (z_{avg} - z_{SZG})/z_{avg}$). Moreover we replicate the minimal solution obtained by our algorithm (z_{min}) and the gap between z_{avg} and z_{min} ($\%gap_{min} = 100 \cdot (z_{avg} - z_{min})/z_{avg}$). With the limited time limits, our algorithm still improves upon the SZG algorithm, leading to better average solution values eight times out of ten, and obtaining an average improvement slightly higher than 9.5%.

7. Conclusions

In this paper, we solved the Bike sharing Rebalancing Problem proposing an effective metaheuristic algorithm in which are implemented a new constructive heuristic and a set of local searches made efficient by new speed-up techniques. A related problem where a maximum duration constraint is considered, the one-commodity Pickup and Delivery Vehicle Routing Problem with maximum Duration, has also been tackled by adapting the developed metaheuristic and an existing branch-and-cut algorithm. The branch-and-cut algorithm considers in an innovative and efficient way the duration constraint, by including inequalities and separation procedures that can be applied to general VRP problems with maximum duration constraints. We evaluated our algorithms on newly collected real-world and literature instances for both problem variants. We strongly improved the solutions reported in the literature getting new optima and new best known solutions in effective computational times.

Acknowledgments

Manuel Iori is partially supported by CAPES/Brazil under Grant PVE no. A007/2013. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a senior research associate, and the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office.

Appendix A

Proof of Property 2. Let us consider each statement one at a time:

a) Let P' be the route obtained by removing vertex i from P. Route P' is feasible if

$$\max_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}-\min_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}\le Q. \tag{A.1}$$

To prove that this is always satisfied, let us consider two cases:

- $q_i \ge 0$. With respect to the original route P, the removal of the vertex may either reduce or keep unchanged both the max and min terms in (A.1). The maximum difference between the two terms arises when the max remains unchanged and the min decreases by the largest possible quantity, i.e., by q_i. We can thus state that:

$$\begin{aligned}&\max_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}-\min_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}\le \max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}\\&\quad-\left(\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-q_i\right)=\max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}\\&\quad+q_i\le \max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}+\Delta_P=Q,\end{aligned}$$

where the last two steps follow, respectively, from the hypothesis and from Eq. (10).

- $q_i < 0$. In this case we consider instead that the largest difference between the two terms in (A.1) is attained when the max increases by $-q_i$ and the min remains unchanged, thus:

$$\begin{aligned}&\max_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}-\min_{k=0}^{|P'|-1}\{l_{P'(k)}(P')\}\le\left(\max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-q_i\right)\\&\quad-\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}=\max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}\\&\quad-q_i\le \max_{k=0}^{|P|-1}\{l_{P(k)}(P)\}-\min_{k=0}^{|P|-1}\{l_{P(k)}(P)\}+\Delta_P=Q.\end{aligned}$$

b) Directly form Property 1 and Property 2.a.

c) Let P' be the route obtained by swapping vertices i and j in P. First suppose that i precedes j in P. Then the proof follows the footsteps of that of point (a), by considering two cases, $q_i \ge q_j$ and $q_i < q_j$, and showing that (A.1) is always satisfied when the hypothesis holds. The same applies when j precedes i.

d) Let P' and R' be the routes obtained from, respectively, P and R, after the swap of i with j. Suppose without loss of generality that $\Delta_P \le \Delta_R$, and let us concentrate on P. Similarly to the proof of point (a), we consider two cases, $q_i \ge q_j$ and $q_i < q_j$. Then one can check that Eq. (A.1) is satisfied on both cases, when the hypothesis holds, by applying a similar consideration to the one above on the largest difference between the max and min values attained in the equation. Note that the fact that the swap is feasible does not imply that the single removal or insertion of a vertex is feasible, i.e., $|q_i|$ and/or $|q_j|$ could be greater than Δ_P. The same reasoning applies to route R and that concludes the proof.□

Proof of Property 3. We are given two disjoint and feasible routes P and R, to be merged in the order $P \oplus R$. We recall that, by using (12), the last forward load window of P can be expressed as

$$\begin{aligned}F_{P(|P|-1)}(P)=\Big[&l_{P(|P|-1)}(P)-\min_{i=0}^{|P|-1}\{l_{P(i)}(P)\},l_{P(|P|-1)}(P)\\&+Q-\max_{i=0}^{|P|-1}\{l_{P(i)}(P)\}\Big].\end{aligned} \tag{A.2}$$

Now notice that the first backward load window of route R can be expressed as the difference between the last forward load window of the route and the cumulated request along the route, that is

$$B_{R(1)}(R)=F_{R(|R|-1)}(R)-l_{R(|R|-1)}(R). \tag{A.3}$$

The extreme values of $F_{R(|R|-1)}(R)$ can be expressed again by using (12), obtaining

$$\begin{aligned}F_{R(|R|-1)}(R)=\Big[&l_{R(|R|-1)}(R)-\min_{i=0}^{|R|-1}\{l_{R(i)}(R)\},l_{R(|R|-1)}(R)\\&+Q-\max_{i=0}^{|R|-1}\{l_{R(i)}(R)\}\Big].\end{aligned} \tag{A.4}$$

Thus combining (A.3) and (A.4) we get

$$B_{R(1)}(R)=\left[-\min_{i=0}^{|R|-1}\{l_{R(i)}(R)\},Q-\max_{i=0}^{|R|-1}\{l_{R(i)}(R)\}\right]. \tag{A.5}$$

Let us define, for the sake of simplicity, $F_{P(|P|-1)}(P)=F=\left[\underline{f},\overline{f}\right]$ and $B_{R(1)}(R)=B=\left[\underline{b},\overline{b}\right]$. Their intersection is given by $F\cap B=\left[\max\left\{\underline{f},\underline{b}\right\},\min\left\{\overline{f},\overline{b}\right\}\right]$.

We first show the "if" part of the thesis, that is, if the intersection of the two load windows is not empty, then the combined route is feasible. The two load windows can be basically seen as two intervals, thus their intersection is not empty if

$$\max\left\{\underline{f},\underline{b}\right\}\le\min\left\{\overline{f},\overline{b}\right\}. \tag{A.6}$$

There are four cases. If $\max\left\{\underline{f},\underline{b}\right\}=\underline{f}$ and $\min\left\{\overline{f},\overline{b}\right\}=\overline{f}$, then (A.6) is satisfied because P is feasible for hypothesis. The same holds when the max and min values are attained by $\underline{b}$ and $\overline{b}$, respectively, because of the feasibility of R. Let us now concentrate on the case where $\max\left\{\underline{f},\underline{b}\right\}=\underline{f}$ and $\min\left\{\overline{f},\overline{b}\right\}=\overline{b}$.

For this case (A.6) summarizes to $\underline{f}\le\overline{b}$, that using (A.2) and (A.5) corresponds to

$$l_{P(|P|-1)}(P)-\min_{i=0}^{|P|-1}\{l_{P(i)}(P)\}\le Q-\max_{i=0}^{|R|-1}\{l_{R(i)}(R)\},$$

which can be rewritten as

$$\max_{i=0}^{|R|-1}\{l_{R(i)}(R)+l_{P(|P|-1)}(P)\}-\min_{i=0}^{|P|-1}\{l_{P(i)}(P)\}\le Q. \tag{A.7}$$

Route $P\oplus R$ is feasible if the following condition is satisfied:

$$\max_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P\oplus R)\}-\min_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P\oplus R)\}\le Q. \tag{A.8}$$

To prove this, first notice that we can rewrite

$$\begin{aligned}\max_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P\oplus R)\}=\max\Big\{&\max_{i=0}^{|P|-1}\{l_{P(i)}(P)\},\\&\max_{i=0}^{|R|-1}\{l_{R(i)}(R)+l_{P(|P|-1)}(P)\}\Big\},\end{aligned} \tag{A.9}$$

$$\begin{aligned}\min_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P\oplus R)\}=\min\Big\{&\min_{i=0}^{|P|-1}\{l_{P(i)}(P)\},\\&\min_{i=0}^{|R|-1}\{l_{P(i)}(R)+l_{P(|P|-1)}(P)\}\Big\}.\end{aligned} \tag{A.10}$$

Because $\overline{f}\ge\overline{b}$, we have that $l_{P(|P|-1)}(P)+Q-\max_{i=0}^{|P|-1}\{l_{P(i)}(P)\}\ge Q-\max_{i=0}^{|R|-1}\{l_{R(i)}(R)\}$, and thus

$$\max_{i=0}^{|R|-1}\{l_{R(i)}(R)+l_{P(|P|-1)}(P)\}\ge\max_{i=0}^{|P|-1}\{l_{P(i)}(P)\},$$

which combined with (A.9) gives

$$\max_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P\oplus R)\}=\max_{i=0}^{|R|-1}\{l_{R(i)}(R)+l_{P(|P|-1)}(P)\}. \tag{A.11}$$

Similarly, from $\underline{f} \geq \underline{b}$, it follows $l_{P(|P|-1)}(P) - \min_{i=0}^{|P|-1}\{l_{P(i)}(P)\} \geq -\min_{i=0}^{|R|-1}\{l_{R(i)}(R)\}$, and hence

$$\min_{i=0}^{|P|-1}\{l_{P(i)}(P)\} \leq \min_{i=0}^{|R|-1}\{l_{R(i)}(R) + l_{P(|P|-1)}(P)\},$$

which combined with (A.10) gives us that

$$\min_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P \oplus R)\} = \min_{i=0}^{|P|-1}\{l_{P(i)}(P)\}. \tag{A.12}$$

By including (A.11) and (A.12) into (A.7) we can thus conclude that (A.8) is satisfied. The remaining case, $\max\{\underline{f}, \underline{b}\} = \underline{b}$ and $\min\{\overline{f}, \overline{b}\} = \overline{f}$, is specular to the one just discussed, and this concludes the first part of the proof.

The "only if" part is proved similarly, noticing that $F \cap B = \varnothing$ means that $\max\{\underline{f}, \underline{b}\} > \min\{\overline{f}, \overline{b}\}$. Once again we have four cases, $\underline{f} > \overline{f}$ and $\underline{b} > \overline{b}$, which are impossible, and $\underline{f} > \overline{b}$ and $\underline{b} > \overline{f}$, which are specular. We concentrate on the case $\underline{f} > \overline{b}$, and use the consequent facts that $\overline{f} \geq \overline{b}$ and $\underline{f} \geq \underline{b}$. By applying similar steps to the ones in the "if" part we can show that

$$\max_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P \oplus R)\} - \min_{i=0}^{|P\oplus R|-1}\{l_{P\oplus R(i)}(P \oplus R)\} > Q,$$

and thus $P \oplus R$ is infeasible.□

References

[1] DeMaio P. Bike sharing: history, impacts, model of provision and future. J Public Transp 2009;10:41–56.
[2] DeMaio P, Meddin R. Bike-sharing world map. [Online]. Available: ⟨http://bike-sharing.blogspot.it/⟩; 2014.
[3] Dell'Amico M, Hadjicostantinou E, Iori M, Novellani S. The bike sharing rebalancing problem: mathematical formulations and benchmark instances. Omega 2014;45:7–19.
[4] Hernández-Pérez H, Salazar-González J-J. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discret Appl Math 2004;145:126–39.
[5] Shi X, Zhao F, Gong Y. Genetic algorithm for the one-commodity pickup-and-delivery vehicle routing problem. In: IEEE international conference on intelligent computing and intelligent systems, vol. 1, 2009. p. 175–9.
[6] Berbeglia G, Cordeau J-F, Gribkovskaia I, Laporte G. Static pickup and delivery problems: a classification scheme and survey. TOP 2007;15:1–31.
[7] Battarra M, Cordeau J-F, Iori M. Pickup and delivery problems for goods transportation. In: Toth P, Vigo D, editors. Vehicle routing: problems, methods, and applications, 2nd ed. SIAM, Monographs on discrete mathematics and applications; 2014, pp 161–192.
[8] Hernández-Pérez H, Salazar-González J-J. The one-commodity pickup-and-delivery travelling salesman problem. Lecture notes in computer science, vol. 2570/2003, Berlin; Springer; 2003. p. 89–104.
[9] Hernández-Pérez H, Salazar-González J-J. The one-commodity pickup-and-delivery traveling salesman problem: inequalities and algorithms. Networks 2007;50:258–72.
[10] Hernández-Pérez H, Salazar-González J-J. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. Transp Sci 2004;38:245–55.
[11] Hernández-Pérez H, Rodríguez-Martín I, Salazar-González J-J. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. Comput Oper Res 2009;36:1639–45.
[12] Martinović G, Aleksi I, Baumgartner A. Single-commodity vehicle routing problem with pickup and delivery service. Math Prob Eng 2008. http://dx.doi.org/10.1155/2008/697981 Article ID 697981.
[13] Zhao F, Li S, Sun J, Mei D. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. Comput Ind Eng 2009;56:1642–8.
[14] Hosny M, Mumford C. Solving the one-commodity pickup and delivery problem using an adaptive hybrid VNS/SA approach. In: Schaefer R, Cotta C, Kolodziej J, Rudolph G, editors. Parallel problem solving from nature, PPSN XI, Lecture notes in computer science, vol. 6239. Berlin; Springer; 2010. p. 189–98.
[15] Mladenović N, Urošević D, Hanafi S, Ilić A. A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. Eur J Oper Res 2012;220:270–85.
[16] Benchimol M, Benchimol P, Chappert B, De La Taille A, Laroche F, Meunier F, et al. Balancing the stations of a self service bike hire system. RAIRO-Oper Res 2011;45:37–61.
[17] Chemla D, Meunier F, Wolfler Calvo R. Bike sharing systems: solving the static rebalancing problem. Discret Optim 2013;10:120–46.
[18] Erdoğan G, Battarra M, Wolfler Calvo R. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. Eur J Oper Res 2015;245:667–79.
[19] Raviv T, Tzur M, Forma I. Static repositioning in a bike-sharing system: models and solution approaches. EURO J Transp Logist 2013;2:187–229.
[20] Papazek P, Raidl G, Rainer-Harbach M, Hu B. A PILOT/VND/GRASP hybrid for the static balancing of public bicycle sharing systems. In: Computer Aided Systems Theory-EUROCAST 2013. Springer; 2013. p. 372–9.
[21] Di Gaspero L, Rendl A, Urli T. Balancing bike sharing systems with constraint programming. Constraints 2015:1–31.
[22] Schuijbroek J, Hampshire R, van Hoeve W-J. Inventory rebalancing and vehicle routing in bike sharing systems. Tepper School of Business, Carnegie Mellon University, Working Paper; 2013.
[23] Contardo C, Morency C, Rousseau L-M. Balancing a dynamic public bike-sharing system, CIRRELT. Technical Report, CIRRELT-2012-09; 2012.
[24] Ergun Ö, Orlin JB. Fast neighborhood search for the single machine total weighted tardiness problem. Oper Res Lett 2006;34:41–5.
[25] Liao C-J, Tsou H-H, Huang K-L. Neighborhood search procedures for single machine tardiness scheduling with sequence-dependent setups. Theor Comput Sci 2012;434:45–52.
[26] Ibaraki T, Imahori S, Kubo M, Masuda T, Uno T, Yagiura M. Effective local search algorithms for routing and scheduling problems with general time-window constraints. Transp Sci 2005;39:206–32.
[27] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp Sci 2006;40:455–72.
[28] Jacobs LW, Brusco MJ. A local search heuristic for large set-covering problems. Nav Res Logist 1995;42:1129–40.
[29] Schrimpf G, Schneider J, Stamm-Wilbrandt H, Dueck G. Record breaking optimization results using the ruin and recreate principle. J Comput Phys 2000;159:139–71.
[30] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher MJ, Puget, J-F, editors. Principles and practice of constraint programming – CP98. Lecture notes in computer science, vol. 1520. Springer, 1998. p. 417–31.
[31] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 2007;177:2033–49.
[32] Richmond AJ, Beasley JE. An iterative construction heuristic for the ore selection problem. J Heuristics 2004;10:153–67.
[33] Cesta A, Oddi A, Smith SF. Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: Proceedings of the seventeenth national conference on artificial intelligence. AAAI Press/The MIT Press; 2000. p. 742–7.
[34] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Oper Res 1964;12:568–81.
[35] Hansen P, Mladenović N. Variable neighborhood search: principles and applications. Eur J Oper Res 2001;130:449–67.
[36] den Besten M, Stützle T. Neighborhoods revisited: an experimental investigation into the effectiveness of variable neighborhood descent for scheduling. In: Proceedings of the 4th metaheuristics international conference, vol. 2, Porto, Portugal; July 2001. p. 545–50.
[37] Dell'Amico M, Iori M, Novellani S, Stützle T. New real-world instances for the bike sharing rebalancing problem. [Online]. Available: ⟨http://www.or.unimore.it/resources/BRP2/instances.html⟩; 2015.
[38] López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M. The irace package, iterated race for automatic algorithm configuration. Technical Report, IRIDIA, Université Libre de Bruxelles, Belgium, TR/IRIDIA/2011-004; 2011. [Online]. Available: ⟨http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf⟩.