Citation for published version:
Bulhões , T, Subramanian, A, Erdoan, G & Laporte, G 2018, 'The static bike relocation problem with multiple vehicles and visits', European Journal of Operational Research, vol. 264, no. 2, pp. 508-523.

Publication date:
2018

Document Version
Early version, also known as pre-print

Link to publication

University of Bath

The static bike relocation problem with multiple vehicles and visits

Teobaldo Bulhões[a], Anand Subramanian[b], Güneş Erdoğan[c], Gilbert Laporte[d]

[a] *Instituto de Computação, Universidade Federal Fluminense*
Rua Passo da Pátria 156, São Domingos, 24210-240, Niterói–RJ, Brazil.
tbulhoes@ic.uff.br
[b] *Departamento de Sistemas de Computação, Centro de Informática, Universidade Federal da Paraíba*
Rua dos Escoteiros, Mangabeira, 58058-600, João Pessoa–PB, Brazil
anand@ci.ufpb.br
[c] *School of Management, University of Bath, BA1 7AY, UK*
g.erdogan@bath.ac.uk
[d] *Canada Research Chair in Distribution Management, HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montreal, Canada H3T 2A7*
gilbert.laporte@cirrelt.ca

Abstract

This paper introduces the *static bike relocation problem with multiple vehicles and visits*, the objective of which is to rebalance at minimum cost the stations of a bike sharing system using a fleet of vehicles. The vehicles have identical capacities and service time limits, and are allowed to visit the stations multiple times. We present an integer programming formulation, implemented under a branch-and-cut scheme, in addition to an iterated local search metaheuristic that employs efficient move evaluation procedures. Results of computational experiments on instances ranging from 20 to 200 stations are provided and analyzed. We also examine the impact of the vehicle capacity and of the number of visits and vehicles on the performance of the proposed algorithms.

Keywords: Routing, Shared mobility systems, Bike sharing, Pickup and delivery

1. Introduction

We study the problem of rebalancing at minimum cost the stations of a *bike sharing system* (BSS) by relocating the bikes using a fleet of vehicles. The inventories of the stations and the travel times between stations are assumed to be static during the rebalancing operation. The fleet is composed of identical vehicles with a given capacity and service time limit, which depart from and return to the *depot*. The cost of the operation is measured as the total travel time of the fleet. The vehicles can visit the stations multiple times, up to a given limit, but a station can only be served by one vehicle of the fleet. In addition to the travel times, we include the handling times of the bikes within the service time limit of the vehicles to ensure that the workload constraint is not violated. This problem is called the *static bicycle relocation problem with multiple vehicles and visits* (SBRP-MVV).

There are now more than 7000 BSSs, as stated in the recent survey by Laporte et al. (2015), and this number is growing at an increasing rate. There exists a body of research on the problem of rebalancing BSSs, mainly on two variants of the problem: the *static* version and the *dynamic* version. The main difference between the two variants is the customer demand during the rebalancing operation, which is assumed to be zero for the static variant while it can be non-zero for the dynamic variant. We refer the interested reader to the papers by Nair and Miller-Hooks (2011), Contardo et al. (2012), and Chemla et al. (2013a) for further details of the dynamic version. In what follows, we provide a brief review of the literature on the static version.

The studies on the static version can be traced back to the seminal paper of Benchimol et al. (2011) in which the authors introduced the *static stations balancing problem* (SSBP) and proved it to be $\mathcal{NP}$-Hard. The SSBP aims at finding a minimum cost route for a single vehicle that can visit the same station multiple times. The vehicle can drop bikes temporarily at intermediate locations for future pickup, i.e. *preemption* was allowed. The studies that followed introduced variants by changing three features: the number of vehicles, the number of visits to stations, and the preemption property. Notably, a few studies have incorporated additional features such as minimizing user dissatisfaction (Raviv et al., 2013), demand intervals for the stations (Erdoğan et al., 2014), and multiple types of bikes (Li et al., 2016). Table 1 provides a summary of the literature on the static bike relocation problems.

Table 1: Literature on static bike relocation problems

		Single vehicle	**Multiple vehicles**
Single visit	Nonpreemptive	Erdoğan et al. (2014) † Ho and Szeto (2014) † ‡ Li et al. (2016) † ‡	Lin and Chou (2012) ‡ Raviv et al. (2013) † Dell'Amico et al. (2014) † Forma et al. (2015) ‡ Dell'Amico et al. (2016) † ‡
Multiple visits	Nonpreemptive	Erdoğan et al. (2015) †	Gaspero et al. (2013) ‡
	Preemptive	Benchimol et al. (2011) † Chemla et al. (2013b) ‡ Erdoğan et al. (2015) † Cruz et al. (2016) ‡	Rainer-Harbach et al. (2014) ‡ Alvarez-Valdes et al. (2016) ‡

† *Exact algorithm*

‡ *Heuristic algorithm*

The reach of the state-of-the-art exact algorithms is 60 stations across the variants of the static bike relocation problem. Table 1 shows that no exact algorithms have been developed for the case of multiple vehicles and multiple visits, a gap we aim to fill with this study by proposing a branch-and-cut algorithm implemented over an integer programming formulation. In addition, we present an iterated local search heuristic that benefits from subsequence based data structures, which allow the algorithm to perform move evaluations in amortized constant time.

The remainder of the paper is organized as follows. Section 2 formally defines the SBRP-MVV and presents an integer programing formulation. Section 3 describes the branch-and-cut algorithm. Section 4 explains the proposed iterated local search meta-heuristic. Section 5 contains the computational experiments. Finally, Section 6 presents the concluding remarks of this work.

2. Mathematical formulation

The SBRP-MVV can be formally defined as follows. We are given a complete and undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Vertex 0 is the depot, while the remaining ones are the bike stations. Each station $i \in V \setminus \{0\}$ has a pickup or a delivery demand q_i. We assume that $q_i > 0$ indicates a pickup demand, whereas $q_i < 0$ denotes a delivery demand. Each edge $\{i, j\} \in E$ has an associated travel time c_{ij}. A fleet of K identical vehicles with capacity Q is available at the depot. A route duration limit L is imposed for each route. There is a service time proportional to the number of bikes to be delivered or collected at a station. More precisely, this service time is given by the product of the handling time h and the number of bikes delivered or collected. Moreover, a station is allowed to be visited at most N times by the same vehicle. We assume that a station cannot be visited by different vehicles, i.e., it cannot appear in two distinct routes in a feasible solution. The objective is to minimize the total travel time.

We define the network $G' = (V', A')$, where $V' = V_C \cup \{0, d\}$. Set V_C represents visits to the stations, and contains N nodes for each station. Vertices 0 and d are the starting and ending points, respectively, for all routes. The arc set A' contains all possible arcs, except those between nodes representing visits to the same station, and therefore there is no arc between 0 and d. Moreover, therefore there is no arc leaving node d or entering node 0.

We define the following notation:

- $\delta^+_{(i)} \subset A'$: the set of arcs leaving vertex $i \in V'$;
- $\delta^-_{(i)} \subset A'$: the set of arcs entering vertex $i \in V'$;

- $\pi(i)$: the vertex in V associated to a vertex $i \in V'$;
- $f(i)$: the vertex $j \in V'$ that represents the first visit to station $i \in V \setminus \{0\}$. We arbitrarily define $j = \min\{u \in V' \mid \pi(u) = i\}$.

The following decision variables are necessary to define the proposed formulation:

- $x_a = 1$ if arc $a \in A'$ is in the solution, and 0 otherwise;
- y_a: the flow of bikes on arc $a \in A'$;
- l_a: the route duration after traversing arc $a \in A'$;
- s_i: the pickup or delivery performed when visiting node $i \in V'$.

The formulation can be written as follows.

$$\text{minimize} \sum_{a \in A'} c_a x_a \tag{1}$$

subject to

$$\sum_{a \in \delta^+_{(i)}} x_a - \sum_{a \in \delta^-_{(i)}} x_a = 0 \qquad i \in V_C \tag{2}$$

$$\sum_{a \in \delta^+_{(0)}} x_a \leq K \tag{3}$$

$$\sum_{a \in \delta^+_{(0)}} x_a = \sum_{a \in \delta^-_{(d)}} x_a \tag{4}$$

$$\sum_{a \in \delta^+_{(i)}} y_a - \sum_{a \in \delta^-_{(i)}} y_a = s_i \qquad i \in V' \tag{5}$$

$$\sum_{j \in V' : \pi(j) = i} s_j = q_i \qquad \forall i \in V \setminus \{0\} \tag{6}$$

$$\sum_{a \in \delta^+_{(i)}} l_a - \sum_{a \in \delta^-_{(i)}} l_a = \sum_{a \in \delta^+_{(i)}} c_a x_a + s_i h \qquad i \in V_C, d_{\pi(i)} > 0 \tag{7}$$

$$\sum_{a \in \delta^+_{(i)}} l_a - \sum_{a \in \delta^-_{(i)}} l_a = \sum_{a \in \delta^+_{(i)}} c_a x_a - s_i h \qquad i \in V_C, d_{\pi(i)} < 0 \tag{8}$$

$$l_a = c_a x_a + h y_a \qquad a \in \delta^+_{(0)} \tag{9}$$

$$l_a + h y_a \leq L x_a \qquad a \in \delta^-_{(d)} \tag{10}$$

$$\sum_{a\in\delta^-_{(i)}} x_a \leq 1 \qquad i \in V_C \tag{11}$$

$$\sum_{a\in\delta^-_{(f(i))}} x_a = 1 \qquad i \in V \setminus \{0\} \tag{12}$$

$$\sum_{a\in S} x_a \leq |S| - 1 \qquad \forall S \subset V' \tag{13}$$

$$l_a \leq L x_a \qquad a \in A' \tag{14}$$

$$y_a \leq Q x_a \qquad a \in A' \tag{15}$$

$$s_0 = \max\{d_0, 0\} \tag{16}$$

$$s_d = \min\{d_0, 0\} \tag{17}$$

$$s_i \geq \sum_{a\in\delta^+_{(i)}} x_a \qquad i \in V_C, d_{\pi(i)} > 0 \tag{18}$$

$$s_i \leq -\sum_{a\in\delta^+_{(i)}} x_a \qquad i \in V_C, d_{\pi(i)} < 0 \tag{19}$$

$$l_a \geq 0 \qquad \forall a \in A' \tag{20}$$

$$y_a \geq 0 \qquad \forall a \in A' \tag{21}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A'. \tag{22}$$

Objective function (1) minimizes the total travel time. Constraints (2) ensure that if there is an arc arriving at a vertex $i \in V_C$ then there should an arc leaving the same vertex i. Constraint (3) guarantees that at most K vehicles leave the depot. Constraint (4) enforces the number of vehicles leaving vertex 0 to be the same as that arriving at vertex d. Constraints (5) ensure the flow conservation of bikes for all vertices of the network. Constraints (6) state that the sum of the deliveries or pickups performed at vertices of the network that represent a visit to a given station i should be equal to the demand of i. Constraints (7) and (8) compute the cumulative travel time after visiting station $i \in V_C$, including the handling time. Constraints (9) and (10) determine the departure and arrival time of a route, respectively, which depend on the number of bikes leaving vertex 0 and arriving at vertex d. Constraints (11) forbid a vertex that represents a visit to a station $i \in V_C$ to be visited more than once. Constraints (12) impose a visit to the vertex that represents the first visit to a station $i \in V \setminus \{0\}$. This vertex can be arbitrarily defined; in our case we selected the vertex with the smallest index among those that represent visits to i. Constraints (13) are subtour inequalities. Constraints (14) limit the duration of a route. Constraints (15) prevent the capacity of the vehicle to be exceeded. Constraints (16) and

(17) determine the delivery and pickup values of vertices 0 and d, respectively. Constraints (18) and (19) state that there should be at least a delivery or a pickup on every visited performed. Constraints (20)–(22) define the domain of the variables.

Formulation (1)–(22) is not complete since it does not prevent a station to be visited by different vehicles. Let $\mathcal{R}$ be the set composed of all pair of routes, represented by their arc sets, with at least one station appearing in both of the them. We thus introduce inequalities (23) to forbid the same station to be visited by distinct vehicles.

$$\sum_{a \in A'} x_a \leq |A'| - 1 \qquad \forall A' \in \mathcal{R}. \tag{23}$$

3. Branch-and-cut algorithm

Since the mathematical formulation described in Section 2 relies on an exponential number of constraints, we have implemented a branch-and-cut (BC) algorithm in order to deal with them in practice. The algorithm initially considers only the polynomial size constraints (2)–(12) and (14)–(22), and then it introduces the exponential constraints (13) and (23) in a dynamic fashion.

Subtour inequalities (13) are separated using a straightforward min-cut based procedure. The BC algorithm tries to separate them only for the nodes whose depth is less than or equal to 10 or whenever an integer solution is found.

Inequalities (23) can be seen as "no-good" cuts, since they are generally useful to ensure feasibility but they do not strengthen the linear relaxation of the formulation. Therefore, such inequalities are only included when an integer infeasible solution is found, that is, in a lazy fashion.

4. Iterated local search metaheuristic

This section describes the metaheuristic we have developed for the SBRP-MVV. The method is mostly based on iterated local search (ILS) (Lourenço et al., 2010), which alternates between local search (intensification) and perturbation (diversification) moves. ILS has been successfully applied to other vehicle routing problems, especially when implemented under a multi-start scheme (Subramanian et al., 2010; Penna et al., 2013; Vidal et al., 2015; Silva et al., 2015).

Figure 1 shows an outline of the developed algorithm. The heuristic performs multiple restarts, where at each of them an ILS based procedure that executes local search and perturbation moves is iteratively called until n_{ILS} consecutive iterations without improvement have been executed. The best solution of the current multi-start iteration is always the one chosen to be perturbed. Initial solutions are generated using an insertion-based constructive heuristic (see Section 4.3) that allows infeasible solutions. However, if a feasible solution is not found after $2n_R$ trials, then the algorithm generates a new initial solution. The algorithm terminates after n_R feasible restarts or $2n_R$ restarts. The local search is executed using a randomized variable neighborhood decent (RVND) procedure (see Section 4.4), whereas the perturbation procedure performs random shift, swap or split moves (see Section 4.5).

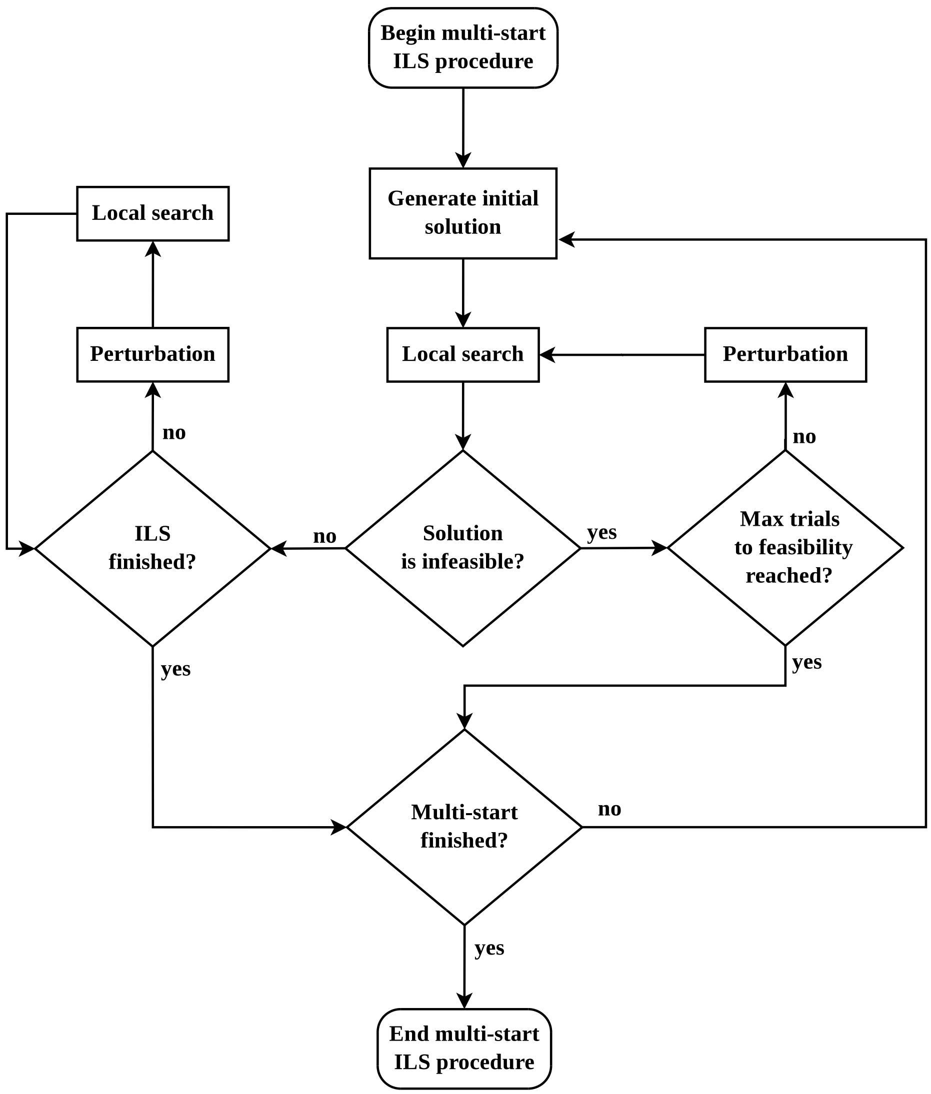


Figure 1: Multi-start ILS flowchart

4.1. Auxiliary data structures

We have implemented some auxiliary data structures (ADSs), following the ideas presented in Hernández-Pérez and Salazar-González (2004), in order to improve the performance of the proposed heuristic both in terms of computational complexity and of the total number of operations performed.

Let $\sigma = (\sigma_{(0)}, \ldots, \sigma_{(|\sigma|-1)})$ be a subsequence of a solution S (with $\overleftarrow{\sigma}$ as the associated reverse subsequence), and let $\sigma_{i,j}$ be the subsequence of σ that starts at the ith position and ends at the jth position, i.e., $\sigma_{i,j} = (\sigma_{(i)}, \ldots, \sigma_{(j)})$. Moreover, let $q'_{\sigma_{(i)}}$ be the load delivered or collected at station $\sigma_{(i)}$ *on that particular visit.* For each possible σ of S the method stores and updates:

- $q_{sum}(\sigma) = \sum_{i=0}^{|\sigma|-1} q'_{\sigma_{(i)}}$ = sum of the loads delivered/collected (cumulative load);
- $q_{min}(\sigma) = \min\{0, q_{sum}(\sigma_{0,0}), q_{sum}(\sigma_{0,1}), \ldots, q_{sum}(\sigma_{0,|\sigma|-1})\}$ = minimum cumulative load;
- $q_{max}(\sigma) = \max\{0, q_{sum}(\sigma_{0,0}), q_{sum}(\sigma_{0,1}), \ldots, q_{sum}(\sigma_{0,|\sigma|-1})\}$ = maximum cumulative load;
- $l_{min}(\sigma) = -q_{min}(\sigma)$ = minimum flow of bikes allowed to enter σ so as to ensure feasibility or so that the infeasibility does not increase;
- $l_{max}(\sigma) = Q - q_{max}(\sigma)$ = maximum flow of bikes allowed to enter σ so as to ensure feasibility or so that the infeasibility does not increase;
- $tt(\sigma) = \sum_{i=0}^{|\sigma|-2} \sum_{j=i+1}^{|\sigma|-1} c_{\sigma_{(i)}\sigma_{(j)}}$ = travel time;
- $dur(\sigma) = tt(\sigma) + h \sum_{i=0}^{|\sigma|-1} |q'_{\sigma_{(i)}}|$ = total duration (travel time + handling time).

Note that it is not necessary to store the ADSs regarding $\overleftarrow{\sigma}$ because they can be directly derived in constant time from those stored for σ:

- $q_{sum}(\overleftarrow{\sigma}) = q_{sum}(\sigma)$;
- $q_{min}(\overleftarrow{\sigma}) = q_{sum}(\sigma) - q_{max}(\sigma)$;
- $q_{max}(\overleftarrow{\sigma}) = q_{sum}(\sigma) - q_{min}(\sigma)$;
- $l_{min}(\overleftarrow{\sigma}) = -q_{sum}(\sigma) + q_{max}(\sigma)$;
- $l_{max}(\overleftarrow{\sigma}) = Q - q_{sum}(\sigma) + q_{min}(\sigma)$;

- $tt(\overleftarrow{\sigma}) = tt(\sigma)$ (assuming that $c_{ij} = c_{ji}$);

- $dur(\overleftarrow{\sigma}) = dur(\sigma)$ (assuming that $c_{ij} = c_{ji}$).

When a subsequence σ' is composed of only one station (or depot) v, then $q_{sum}(\sigma') = q'_v$, $q_{min}(\sigma') = \min(0, q'_v)$,$q_{max}(\sigma') = \max(0, q'_v)$, $l_{min}(\sigma') = -q_{min}(\sigma')$, $l_{max}(\sigma') = Q - q_{max}(\sigma')$, $tt(\sigma') = 0$ and $dur(\sigma') = h|q'_v|$. Let the operator $\oplus$ denote a concatenation between two distinct subsequences. In what follows we show that any subsequence σ, $|\sigma| > 1$, can be derived from two other subsequences σ^1 and σ^2 by means of the concatenation operator $\oplus$:

$$q_{sum}(\sigma^1 \oplus \sigma^2) = q_{sum}(\sigma^1) + q_{sum}(\sigma^2); \tag{24}$$

$$q_{min}(\sigma^1 \oplus \sigma^2) = \min\{q_{min}(\sigma^1), q_{sum}(\sigma^1) + q_{min}(\sigma^2)\}; \tag{25}$$

$$q_{max}(\sigma^1 \oplus \sigma^2) = \max\{q_{max}(\sigma^1), q_{sum}(\sigma^1) + q_{max}(\sigma^2)\} \tag{26}$$

$$l_{min}(\sigma^1 \oplus \sigma^2) = -q_{min}(\sigma^1 \oplus \sigma^2); \tag{27}$$

$$l_{max}(\sigma^1 \oplus \sigma^2) = Q - q_{max}(\sigma^1 \oplus \sigma^2); \tag{28}$$

$$tt(\sigma^1 \oplus \sigma^2) = tt(\sigma^1) + c_{\sigma^1_{(|\sigma^1|-1)}\sigma^2_{(0)}} + tt(\sigma^2); \tag{29}$$

$$dur(\sigma^1 \oplus \sigma^2) = dur(\sigma^1) + c_{\sigma^1_{(|\sigma^1|-1)}\sigma^2_{(0)}} + dur(\sigma^2). \tag{30}$$

The total number of subsequences of a solution S is of the order of $|V|^2$. Since the information stored for each subsequence can be updated in constant time, it takes $\mathcal{O}(|V|^2)$ operations to update all ADSs.

We now provide an example. Let $\sigma = (2, 1, 3, 4, 1)$ be a subsequence involving 4 stations (1, 2, 3 and 4) and 5 visits with $q'_{\sigma_{(0)}} = 3$, $q'_{\sigma_{(1)}} = -3$, $q'_{\sigma_{(2)}} = 4$, $q'_{\sigma_{(3)}} = -2$, and $q'_{\sigma_{(4)}} = -1$. The vehicle collects three bikes at station 2, delivers three bikes at station 1, collects four bikes at station 3, delivers two bikes at station 4, and finally delivers one bike at station 1. Note that station 1 is visited twice. Moreover, let us assume that: $Q = 5$, $h = 2$, $c_{21} = 2$, $c_{13} = 1$, $c_{34} = 3$ and $c_{41} = 2$. We will thus have the following values for the ADSs for this subsequence:

- $q_{sum}(\sigma) = 3 - 3 + 4 - 2 - 1 = 1$;

- $q_{min}(\sigma) = \min(0, 3, 0, 4, 2, 1) = 0$;

- $q_{max}(\sigma) = \max(0, 3, 0, 4, 2, 1) = 4$;

- $l_{min}(\sigma) = 0$;

- $l_{max}(\sigma) = 5 - 4 = 1$;

- $tt(\sigma) = 2 + 1 + 3 + 2 = 8$;
- $dur(\sigma) = 8 + 2(3 + 3 + 4 + 2 + 1) = 34$.

Note that the number of bikes that can enter σ, so as to ensure feasibility, must lie within the interval $[l_{min}(\sigma), l_{max}(\sigma)] = [0, 1]$. Suppose now that $Q = 3$. This leads to an infeasibility ($[l_{min}(\sigma), l_{max}(\sigma)] = [0, -1]$) since the capacity of the vehicle would be exceeded when visiting station 3.

4.2. Evaluation function

Let w_Q, w_L and w_N be the penalty weights associated with violations on load, route duration and number of visits, respectively. We also define $V(\sigma)$ as the set of stations that are part of a route σ, and n_i as the number of visits to a station $i \in V(\sigma)$. The evaluation function of route σ, which can be seen as a subsequence starting and ending at the depot, is given by

$$Z(\sigma) = tt(\sigma) + w_Q(\max\{0, -q_{min}(\sigma)\} + \max\{0, q_{max}(\sigma) - Q\}) + w_L(\max\{0, dur(\sigma) - L\}) + w_N \sum_{i \in V(\sigma)} (\max\{0, n_i - N\}). \tag{31}$$

The first term of the right-hand side of Equation (31) measures the travel time of the route. The second one computes the penalty regarding the maximum violation on the vehicle load. The violation occurs when the load exceeds the vehicle capacity or when the load becomes negative. The third term is the penalty incurred when the route duration exceeds the maximum limit. Finally, the last term computes the penalty in case of violation on the maximum number of visits.

The values of w_Q, w_L and w_N are initially set to 1000, 10 and 100, respectively. If the local search returns an infeasible solution, then the penalty coefficients are automatically adjusted as follows:

- in case of infeasibility due to load: $w_Q = \min\{100000, 1.2w_Q\}$; otherwise, $w_Q = \max\{1000, 0.8w_Q\}$;
- in case of infeasibility due to route duration: $w_L = \min\{1000, 1.2w_Q\}$; otherwise, $w_L = \max\{10, 0.8w_Q\}$;
- in case of infeasibility due to visits, $w_N = \min\{10000, 1.2w_Q\}$; otherwise, $w_N = \max\{100, 0.8w_Q\}$.

4.3. Constructive procedure

Algorithm 1 describes the insertion procedure used to build an initial solution. Let R be the set composed of K initially empty routes and let SL be a set composed of the stations whose demands have not been fully met in the partial solution. Firstly, all stations are considered to be part of SL (lines 7–8). We define S as the partial initial solution and CL as the candidate list composed of tuples containing information regarding each insertion. A tuple has the format (v, p, r, Δ, q', δ_0), where v is the station, p is the position in which v would be inserted in a given route r, Δ is the insertion cost of v in the associated position p and route r, q' is the load to be delivered ($q' < 0$) or collected ($q' > 0$) in v, and δ is the variation on the number of bikes at the beginning of the route. In addition, let $rd(v)$ be the residual demand of station v and let $rd(0)$ be the residual demand of the depot.

While SL is not empty, the candidate list of all possible insertions is built at every iteration (lines 10-30). The load q' to be delivered or collected during a particular visit is determined by an auxiliary procedure called `DecideStationLoad`. This procedure not only specifies the value of q', but also the value of δ (line 14). In a first round, the algorithm only considers feasible insertions (lines 15–16). However, when this is no longer possible (lines 20–22), the total residual demand of a station is met in a single yet infeasible visit (lines 17–19). Next, the insertion cost is computed and CL is updated (lines 18–19). Note that a station is only allowed to be inserted in route r in case it has not yet been added to the partial solution S or it has been already added to route r itself. In practice, this is to prevent the same station from being visited by distinct routes.

Once CL is built, one element is selected using the same idea of the construction phase of the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic (Resende and Ribeiro, 2010) (line 23). More precisely, a restricted candidate list (RCL) is created by choosing the elements from CL associated with the best insertions. The size of RCL is controlled by a parameter α that defines the level of greediness or randomness. The larger the value of α the larger the size of RCL. In our experiments α is selected at random from the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. An element is then randomly chosen from RCL and the associated station v is inserted into the partial solution S on the specified position p of route r with the corresponding load q' (line 24). The initial load of route $g(r)$ as well as the load of the depot are updated in lines 25–26. Next, the residual demand of station v is updated and, in case it turns out to be zero, v is removed from SL (lines 27–29). Moreover, all ADSs are updated accordingly (line 30).

After all vertices have been inserted and their respective demands are fully met, the algorithm checks whether a pair of identical vertices appear next to each other in a route.

Algorithm 1 Constructive procedure

1: **procedure** BuildInitialSolution(seed, data)
2: $SL \leftarrow V \setminus \{0\}$
3: $S \leftarrow \emptyset$
4: $firstRound \leftarrow true$
5: $l_{depot} \leftarrow \max(0, rd(0))$ /**Load of the depot**/
6: $l_{ini}(r) \leftarrow 0, \ \forall r \in R$ /**Initial load of route r**/
7: **for** each station $v \in V \setminus \{0\}$ **do**
8:     $SL \leftarrow v$
9: **while** $|SL| > 0$ **do**
10:     $CL \leftarrow \emptyset$
11:     **for** each $v \in SL$ **do**
12:         **for** each $r \in R$ in which v can be inserted **do**
13:             **for** each position p of route r **do**
14:                 $[q', \delta] \leftarrow$ DecideStationLoad$(v, p, r, rd(v), l_{ini}(r), l_{depot})$ /**See Alg. 2**/
15:                 **if** $q' = 0$ **and** $firstRound = false$ **then**
16:                     $q' \leftarrow rd(v)$
17:                 **if** $q' \neq 0$ **then**
18:                     $\Delta \leftarrow$ cost of inserting v in position p of r
19:                     $CL \leftarrow (v, p, r, \Delta, \delta, q')$
20:     **if** $|CL| = 0$ **then**
21:         $firstRound \leftarrow false$
22:         Go to line 9
23:     $g \leftarrow$ element from CL selected using the idea of the constructive phase of GRASP
24:     $S \leftarrow S \cup g(v)$ in position $g(p)$ of route $g(r)$ with load $g(q')$
25:     $l_{ini}(g(r)) \leftarrow l_{ini}(g(r)) + g(\delta)$
26:     $l_{depot} \leftarrow l_{depot} - g(\delta)$
27:     $rd(v) \leftarrow rd(v) - g(q')$
28:     **if** $rd(v) = 0$ **then**
29:         $SL \leftarrow SL \setminus \{v\}$
30:     Update ADSs
31: Check for consecutive visits to the same station in a route and, if it is the case, merge them so that only a single visit is performed
32: **return** S
33: **end** BuildInitialSolution

If so, these are merged into a single visit (line 31). Finally, the initial solution is returned (line 32).

Algorithm 2 presents the auxiliary procedure employed to decide the number of bikes to be delivered or collected at station v when considering an insertion in a possible position p of route r. The decision is performed based (i) on the cumulative load and load intervals of the subsequence of r starting from the first visit and ending at position $p-1$, here denoted as subsequence σ^1; (ii) on the load intervals of the subsequence of r starting from the station associated with position p and ending at the last visit, here denoted as subsequence σ^2; (iii) on the extra load initially available at the depot (l^+) that may be used to meet the residual demand of a delivery station v without violating the loading constraints of σ^1; (iv) on the extra number of bikes that may be brought back to the depot (l^-) due to the insertion of a pickup station v. Note that these informations can be accessed in constant time by simply checking the values of $q_{sum}(\sigma^1)$, $l_{min}(\sigma^1)$, $l_{min}(\sigma^2)$ and $l_{max}(\sigma^2)$.

Algorithm 2 Decide the load of a station to be inserted in position p of a route r

1: **procedure** DecideStationLoad($v, p, r, rd(v), l_{ini}(r), l_{depot}$)
2: Let σ^1 be the subsequence of r starting from the first visit of a the route r and ending at the station associated with position $p-1$
3: Let σ^2 be the subsequence of r starting from the station associated with position p and ending at the last visit of route r
4: $q' \leftarrow 0$; $\delta \leftarrow 0$
5: **if** $rd(v) < 0$ **then**
6: $\quad l^+ \leftarrow max(0, min(l_{depot}, l_{max}(\sigma^1) - l_{ini}(r)))$ /**Extra load available at the depot that does not violate* σ^1*/
7: $\quad$ **if** $q_{sum}(\sigma^1) + l_{ini}(r) + l^+ > l_{min}(\sigma^2)$ **then**
8: $\quad\quad q' \leftarrow -min(q_{sum}(\sigma^1) + l_{ini}(r) + l^+ - l_{min}(\sigma^2), |rd(v)|)$
9: $\quad\quad$ **if** $|q'| > max(0, q_{sum}(\sigma^1) + l_{ini}(r) - l_{min}(\sigma^2))$ **then**
10: $\quad\quad\quad \delta \leftarrow |q'| - max(0, q_{sum}(\sigma^1) + l_{ini}(r) - l_{min}(\sigma^2))$
11: **else**
12: $\quad l^- \leftarrow max(0, l_{ini}(r) - l_{min}(\sigma^1)))$ /**Extra load returning to the depot**/
13: $\quad$ **if** $q_{sum}(\sigma^1) + l_{ini}(r) - l^- < l_{max}(\sigma^2)$ **then**
14: $\quad\quad q' \leftarrow min(l_{max}(\sigma^2) - (q_{sum}(\sigma^1) + l_{ini}(r) - l^-), rd(v))$
15: $\quad\quad$ **if** $q' > max(0, l_{max}(\sigma^2) - (q_{sum}(\sigma^1) + l_{ini}(r)))$ **then**
16: $\quad\quad\quad \delta \leftarrow max(0, l_{max}(\sigma^2) - (q_{sum}(\sigma^1) + l_{ini}(r))) - q'$
17: **return** $[q', \delta]$
18: **end** DecideStationLoad

If a vehicle leaving σ^1 enters σ^2 with more bikes than the minimum limit required by σ^2, i.e, $q_{sum}(\sigma^1)+l_{ini}(r)+l^+ > l_{min}(\sigma^2)$, it is then possible to insert a delivery station v ($rd(v) < 0$) in position p with a corresponding load q' given by the maximum between the excess of bikes of the vehicle and $rd(v)$ (lines 5–10). Analogously, if a vehicle leaving σ^1 enters σ^2 with fewer bikes than the maximum limit required by σ^2, i.e, $q_{sum}(\sigma^1)+l_{ini}(r)-l^- < l_{max}(\sigma^2)$, it is then possible to insert a pickup station v ($rd(v) > 0$) in position p with a corresponding load q' given by the minimum between the residual capacity of the vehicle and $rd(v)$ (lines 11–16). The variation on the depot load is computed in lines 9–10 and 15–16.

4.4. Local search

RVND (Subramanian et al., 2010; Subramanian, 2012) extends the well-known VND procedure (Mladenović and Hansen, 1997) by allowing a random ordering of the neighborhoods. All possible moves of each neighborhood is examined and the search continues from the best improving neighbor. If a neighborhood is not capable of finding an improved solution, the procedure then selects another one at random.

We consider two classes of neighborhood structures: inter-route and intra-route. The first performs moves between a pair of routes, whereas the latter only consider moves within the same route. The main RVND scheme considers only inter-route neighborhoods. Intra-route neighborhoods are only applied, also in an RVND fashion, over those routes that have been affected by an inter-route move or by a perturbation move.

As in the construction phase, if the local search procedure detects that there are two or more consecutive visits to the same station in a modified solution, they are then merged into a single visit. It should be also pointed out that the ADSs are only updated for the subsequences that were modified by a move. Finally, all moves are evaluated in amortize constant time by using the information stored in the ADSs.

4.4.1. Inter-route neighborhood structures

The following inter-route neighborhood structures were implemented:

- Shift($\sigma, 0$): a subsequence σ is moved from one route to another one. We have limited the size of σ to 1 and 2. Each size is assumed to be a different neighborhood in the RVND. Only those subsequences that do not have stations visited multiple times in the route are considered.

- Swap(σ^1, σ^2): subsequence σ^1 from one route is interchanged with another subsequence σ^2 from a different route. We have limited the size of σ^1 and σ^2 to 1 and 2. Disregarding symmetries, each of the three possibilities is assumed to be a different neighborhood in the RVND. As in the previous case, only those subsequences containing stations with a single visit in the respective route are considered.

- 2-opt*: two distinct routes are divided into two subsequences each: σ^1 and σ^2, for the first route, and σ^3 and σ^4, for the second one. Next, two new routes are derived by connecting σ^1 with σ^4 and σ^3 with σ^2.

4.4.2. Intra-route neighborhood structures

The following intra-route neighborhood structures were implemented:

- Reinsertion(σ): a subsequence σ is removed and reinserted in another position of the route. The size of σ was limited to 1, 2 and 3, thus resulting in three different neighborhoods.

- Exchange(σ^1, σ^2): subsequence σ^1 is interchanged with other subsequence σ^2. The size of σ^1 and σ^2 were restricted to 1 and 2, thus leading to three distinct neighborhoods (disregarding symmetries).

- 2-opt: an arc is removed and another one is inserted so as to build a new route.

- Split: a visit is selected and then a copy of the station associated with such visit is inserted in another position (non-adjacent to the original visit) of the route. All split possibilities are considered.

4.5. Perturbation mechanisms

The perturbation procedure randomly selects one of the following mechanisms:

- Multiple shift($\sigma, 0$): multiple Shift($\sigma, 0$) moves are applied at random and we limited $|\sigma|$ to be at most 3.

- Multiple swap(σ^1, σ^2): multiple Swap(σ^1, σ^2) moves are applied at random. In our case we limited $|\sigma^1| = |\sigma^2|$ to be at most 3.

- Split in half: this is a particular case of the Split neighborhood, where the value of the original load is equally split between the original visit and the copy, and the latter is inserted in a random position (non-adjacent to the original visit) of the route. In this case the mechanism selects the visit associated with the largest delivery/pickup of a random route.

5. Computational Experiments

The BC algorithm was implemented using CPLEX 12.6 over the mathematical formulation described in Section 2. This algorithm and the ILS based heuristic presented in Section 4 were both coded in C++. BC was executed on a Intel Xeon E5-2650 v2 processor with a clock speed of 2.60 GHz and 64 GB of RAM, running on Scientific Linux release 6.5 (Carbon), while ILS was executed on an Intel i7-3770 with 3.40 GHz and 16 GB of RAM running Ubuntu 14.04. We considered only a single thread when running the algorithms. ILS was run 10 times for each instance.

5.1. Instances

The SBRP-MVV instances were derived from those proposed by Hernández-Pérez and Salazar-González (2004) for the one-commodity pickup and delivery traveling salesman problem (1-PDTSP), which are available at http://hhperez.webs.ull.es/PDsite/#Benchmark. We considered a total of 630 instances involving up to 200 vertices and, for each of them, we adopted two distinct values for the number of vehicles and for the maximum number of visits allowed, namely, $K = \{2, 3\}$ and $N = \{2, 3\}$. The route duration limit for each instance was computed as follows: $L = \lceil f \times \overline{LB}/K \rceil$, where $\overline{LB}$ is computed by solving the LP relaxation of the model SBRP-R of Erdoğan et al. (2015) and f is the minimum value in the set $\{1, 1.1, 1.2, 1.3, \ldots\}$ for which an "aggressive" version of the heuristic is capable of finding a feasible solution.

Preliminary experiments revealed that a station is seldom visited more than twice, even for those instances with $Q = 10$. Therefore, we decided to disregard the instances with $N = 3$, leading to a total of $2 \times 630 = 1260$ instances.

5.2. Parameter tuning

Regarding the parameter n_{ILS}, we set its value as a function of the instance, more precisely, $n_{ILS} = \max(I_{min}, n)$, where I_{min} is an input parameter used to avoid an insufficient number of perturbations for small size instances. We then followed the procedure described in Cruz et al. (2016) to calibrate the parameter I_{min}. For this testing we arbitrarily set $n_R = 1$ and we adopted $|\sigma| = 1$ for neighborhoods Shift(σ,0) and Reinsertion(σ), $|\sigma^1| = |\sigma^2| = 1$ for neighborhood Swap(σ^1, σ^2), and $|\sigma^1| = |\sigma^2| \leq 2$ for neighborhood Exchange(σ^1, σ^2). After performing some experiments on a subset of 30 random instances, containing 2 instances with capacity values in $\{10, 15, 20\}$ for each size in $\{20, 40, 60, 100, 200\}$, we set $I_{min} = 100$. We chose instances with tight capacity values because they are likely to be more harder to solve.

Once we set $n_{ILS} = \max\{100, n\}$, we performed a series of experiments to calibrate the parameter n_R, and to determine the neighborhood structures to be used in the local search. We tested three different values for n_R and for each of them we tried four different types of neighborhood combinations. Note that because there are 30 instances and two values for K, then each scenario contains $30 \times 2 = 60$ instances. Since we executed the algorithm 10 times for each setting, the total number of runs for each setting is equal to 600. The percentage of feasible solutions returned by the algorithm in the 600 runs for each setting, as well as the average gap between the average solutions and the best solution found during the tuning phase are reported in Table 2. The latter was computed considering only those instances where all settings could find feasible solutions in all 10 runs. From the results obtained, we decided to set $n_R = 20$ and the neighborhoods selected were Shift(1,1), Swap(1,1), 2-opt*, Reinsertion(1), Exchange(1,1), Exchange(2,2), 2-opt and Split.

Table 2: Percentage of feasible runs and average solution cost (out of 600) for each setting

Local Search Configuration	$n_R = 10$		$n_R = 15$		$n_R = 20$	
	Feas (%)	Avg Gap (%)	Feas (%)	Avg Gap (%)	Feas (%)	Avg Gap (%)
Shift(1,1) + Swap(1,1) + 2-opt* Reinsertion(1) + Exchange(1,1) + 2-opt + Split	71.17	0.82	83.00	0.51	90.33	0.33
Shift(1,1) + Swap(1,1) + 2-opt* Reinsertion(1) + Exchange(1,1) + 2-opt + Split + Reinsertion(2,2)	72.00	0.69	86.50	0.50	92.83	0.26
Shift(1,1) + Swap(1,1) + 2-opt* Reinsertion(1) + Exchange(1,1) + 2-opt + Split + Exchange(2,2)	75.00	0.78	85.33	0.43	92.00	0.31
Shift(1,1) + Swap(1,1) + 2-opt* + Reinsertion(1) + Exchange(1,1) + 2-opt + Reinsertion(2,2) + Exchange(2,2) + Split	73.83	0.72	89.33	0.36	**92.67**	**0.18**

5.3. Evaluating the impact of Q, N and K on solving the SBRP-MVV

In this section we are interested in evaluating the impact of the vehicle capacity and number of vehicles on the number of visits to a station, as well as on the average gap. We also examine the benefits and disadvantages of allowing multiple visits, in terms of solution quality and CPU time, according to the vehicle capacity.

Figure 2 shows the average percentage of instances for which the best solution found by ILS required multiple visits to the same station according to the value of Q. We can verify that the percentage of instances requiring multiple visits tends to increase as the capacity of the vehicle decreases. This observation becomes more evident for $Q = \{10, 15, 20\}$. Moreover, it seems that multiple visits are more frequent for $K = 2$ than for $K = 3$.

Figure 3 illustrates the average gap between the best solutions found by ILS and the lower bounds obtained by BC for the different values of Q. It shows that the smaller the capacity, the harder the instance. Also, the instances appear to become harder when there are more vehicles available. However, it is important to point out that the value of L decreases as the number of vehicle increases (see Section 5.1), which may potentially contribute to increase the level of difficulty of solving the SBRP-MVV.

From the results presented in Figures 2 and 3 it is possible to verify that multiple visits seem to be less attractive for $Q \geq 25$. We thus hereafter only report the results for the instances with $Q \leq 20$.

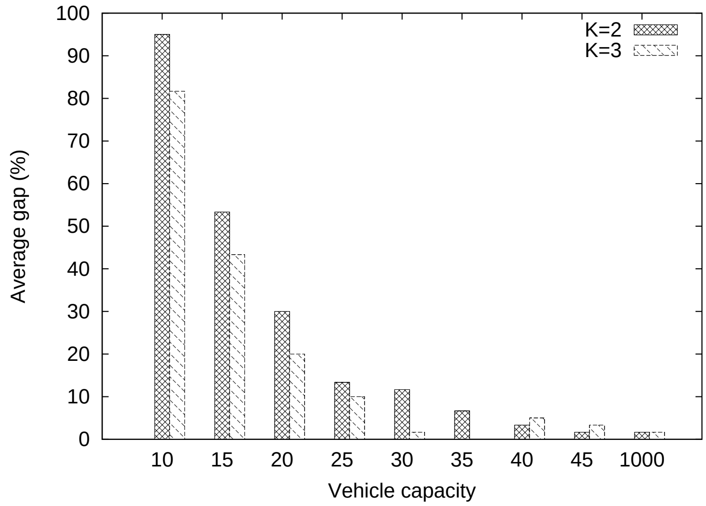


Figure 2: Average percentage of instances for which the best solution found required multiple visits to the same station

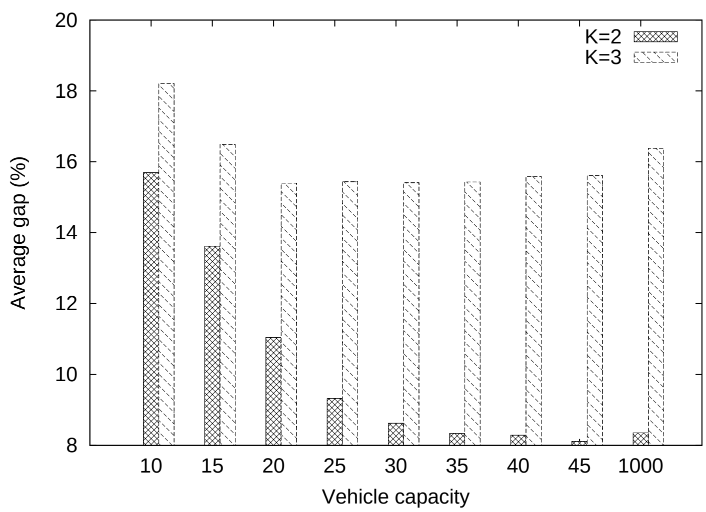


Figure 3: Average gaps between the best solution found by ILS and the lower bound obtained by BC

Figure 4 depicts the average percentage improvement on the best solution obtained by allowing multiple visits. Our aim in this case is to estimate the benefits of letting a station to be visited more than once on the quality of the best solution found by ILS as the vehicle capacity increases. The results suggest that the improvement is considerable for $Q = 10$ and still significant for $Q = 15$ and $Q = 20$, but only for $K = 2$ in the latter. In addition, they are also in accordance with the results shown in Figure 2, that is, we can see that the number of vehicles has an impact on the necessity of multiple visits for finding high quality solutions.

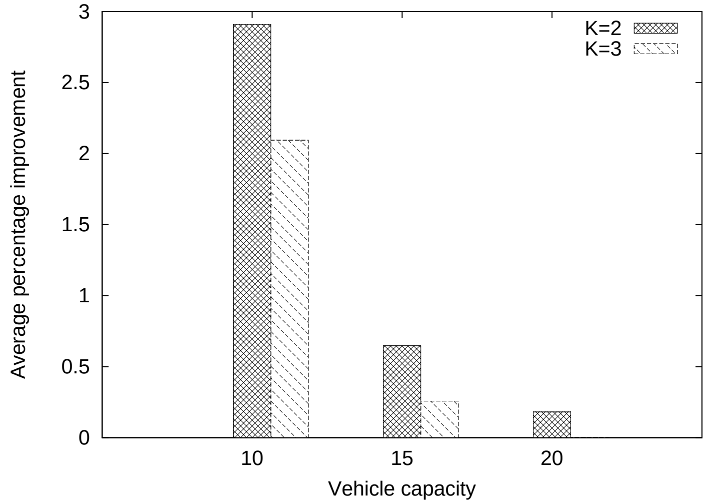
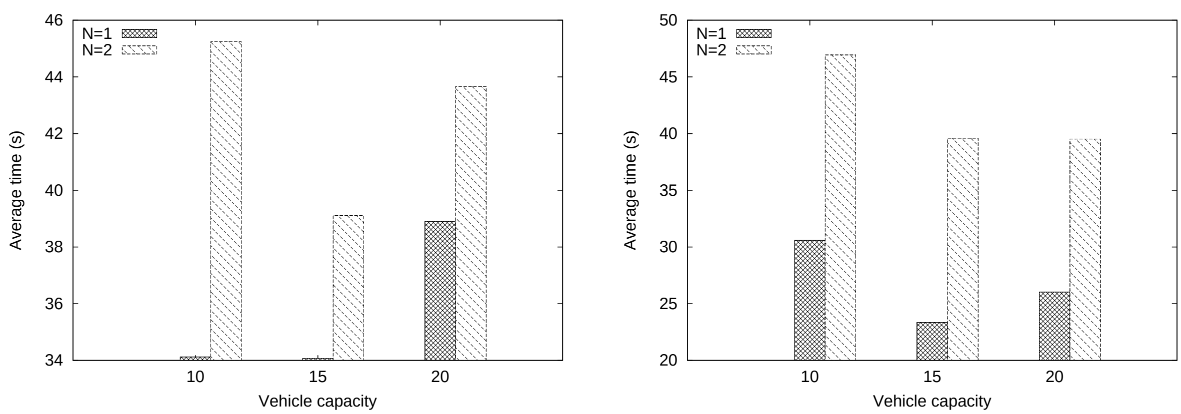


Figure 4: Average percentage improvement on the best solution obtained by allowing multiple visits

Finally, Figure 5 shows the impact on CPU time of ILS when allowing multiple visits to a station. In this case there is a clear runtime disadvantage when the algorithm tries to exploit the possibility of visiting stations more than once. Unfortunately, this visibly affects the convergence rate, especially for $K = 3$, where the increase in the CPU time is more perceptible.



(a) $K = 2$ (b) $K = 3$

Figure 5: Average CPU time spent by ILS

5.4. Detailed results for instances with 20 and 30 stations

Tables 3–6 present the detailed results found by the BC and ILS algorithms for the instances involving 20 and 30 stations. Regarding the BC results, **Root LB** denotes the lower bound obtained after solving the root node, **LB** corresponds to the best lower bound found, **Root time (s)** corresponds to the CPU time in seconds spent to solve the root node, **Time (s)** is the total CPU time in seconds, **Tree size** indicates the number of nodes of the BC tree and **#Lazy cuts** reports the number of lazy cuts (23) added. For what concerns the ILS results, **Min cost** denotes the best cost found in the 10 runs, **Avg Cost** corresponds to the average cost of the 10 runs, **Avg Gap (%)** indicates the average gap between the average solution and the lower bound, and **Avg Time (s)** represents the average CPU time of the 10 runs.

Table 3: Summary of computational results for $n = 20$, $K = 2$ and $N = 2$

		BC						ILS			
Instance	L	Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n20q10A	3192	4274.06	4463.36	9.86	3600.00	173600	2	5006	5006.00	12.16	0.58
n20q10B	3342	4492.06	5052.00	5.82	3250.87	253164	92	**5052**	5052.00	0.00	0.61
n20q10C	4134	5600.29	5697.99	23.20	3600.00	94884	3	6118	6118.00	7.37	1.00
n20q10D	3893	5658.83	5848.94	13.12	3600.00	112451	0	6277	6277.00	7.32	0.83
n20q10E	3939	5761.67	5889.45	15.99	3600.00	104090	0	6381	6381.00	8.35	0.89
n20q10F	3236	4575.17	4907.33	9.91	3600.00	123061	2	4983	4983.00	1.54	0.80
n20q10G	3387	4694.64	4994.90	7.76	3600.00	156040	0	5788	5788.00	15.88	0.59
n20q10H	3727	4957.18	5263.46	9.79	3600.00	155817	5	6328	6328.00	20.23	0.62
n20q10I	3218	4222.47	4428.70	8.36	3600.00	179674	0	4979	4979.00	12.43	0.64
n20q10J	3160	4136.95	4317.88	5.50	3600.00	265948	153	4995	4995.00	15.68	0.69
n20q15A	2736	3770.51	4047.56	6.48	3600.00	194713	1	4435	4435.00	9.57	0.49
n20q15B	3052	4284.43	4740.00	6.80	1011.28	80797	1	**4740**	4740.00	0.00	0.49
n20q15C	3528	4691.84	4860.09	11.74	3600.00	117417	3	5494	5494.00	13.04	0.79
n20q15D	3743	4769.17	4896.68	16.11	3600.00	120330	0	5638	5638.00	15.14	1.12
n20q15E	3611	4962.57	5240.24	12.45	3600.00	137366	3	5800	5800.00	10.68	0.65
n20q15F	2998	4406.85	4645.54	7.94	3600.00	120407	8	4923	4923.00	5.97	0.76
n20q15G	3233	4431.96	4707.58	5.82	3600.00	176153	0	5218	5218.00	10.84	0.53
n20q15H	3558	4567.60	5188.32	9.41	3600.00	114336	308	5635	5635.00	8.61	0.59
n20q15I	3058	3985.14	4316.49	6.01	3600.00	174000	5	4615	4615.00	6.92	0.67
n20q15J	3028	3840.26	4125.24	5.78	3600.00	250276	49	4270	4270.00	3.51	0.54
n20q20A	2736	3642.39	3947.75	3.12	3600.00	248072	0	4435	4435.00	12.34	0.44
n20q20B	3052	4091.14	4740.00	4.30	2435.69	229683	0	**4740**	4740.00	0.00	0.55
n20q20C	3267	4359.78	4605.94	10.17	3600.00	115222	0	5203	5203.00	12.96	0.63
n20q20D	3294	4217.71	4440.28	9.97	3600.00	142800	1	5016	5016.00	12.97	0.69
n20q20E	3282	4561.37	4763.00	6.44	56.32	1438	0	**4763**	4763.00	0.00	0.52
n20q20F	3062	4348.20	4674.00	6.95	124.99	4639	0	**4674**	4674.00	0.00	0.57
n20q20G	3079	4361.35	4798.13	6.27	3600.00	143011	0	5256	5256.00	9.54	0.62
n20q20H	3388	4339.99	4937.48	4.97	3600.00	189500	86	5320	5320.00	7.75	0.57
n20q20I	2897	3995.80	4272.24	4.98	3600.00	200668	0	4640	4640.00	8.61	0.65
n20q20J	3028	3713.55	4145.00	5.16	269.76	12342	0	**4145**	4145.00	0.00	0.44

Table 4: Summary of computational results for $n = 30$, $K = 2$ and $N = 2$

Instance	L	BC						ILS			
		Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n30q10A	4114	5742.16	5796.06	49.04	3600.00	27304	0	6552	6559.50	13.17	2.18
n30q10B	4129	5858.21	5933.98	41.60	3600.00	31947	0	6617	6617.00	11.51	1.46
n30q10C	4292	5618.32	5668.99	45.29	3600.00	36139	0	6625	6625.40	16.87	3.45
n30q10D	4354	5446.28	5570.54	35.60	3600.00	39003	0	6219	6222.40	11.70	2.05
n30q10E	4023	5295.69	5411.80	35.39	3600.00	36225	0	6387	6387.00	18.02	1.74
n30q10F	3885	5214.95	5292.49	51.77	3600.00	44699	0	5977	5977.00	12.93	2.10
n30q10G	5605	7995.47	8070.85	62.21	3600.00	30714	0	9109	9112.00	12.90	2.37
n30q10H	4222	5350.89	5418.30	67.71	3600.00	32729	0	6138	6162.90	13.74	2.27
n30q10I	3752	4899.13	4961.84	35.27	3600.00	38162	0	5764	5764.00	16.17	1.89
n30q10J	4194	5471.28	5542.50	30.30	3600.00	37700	0	6026	6026.00	8.72	2.08
n30q15A	3723	5060.29	5159.71	34.49	3600.00	30044	0	5858	5858.00	13.53	1.69
n30q15B	3653	5023.00	5122.76	35.91	3600.00	51226	0	5682	5682.00	10.92	1.15
n30q15C	3777	4956.43	5093.17	18.79	3600.00	40921	0	5636	5636.00	10.66	2.84
n30q15D	3684	4912.23	5037.13	22.69	3600.00	42170	0	5844	5844.00	16.02	1.62
n30q15E	3713	4796.80	4931.23	21.79	3600.00	54917	0	5803	5803.00	17.68	1.62
n30q15F	3437	4722.68	4866.74	28.61	3600.00	52531	0	5388	5388.00	10.71	1.74
n30q15G	4701	6345.69	6457.06	46.49	3600.00	38018	0	7623	7648.20	18.45	2.54
n30q15H	3597	4570.93	4669.42	45.72	3600.00	34215	0	5304	5304.00	13.59	1.39
n30q15I	3262	4460.33	4602.80	24.55	3600.00	40396	2	4929	4929.00	7.09	1.07
n30q15J	3728	4895.24	4962.95	37.55	3600.00	29663	0	5847	5847.00	17.81	1.49
n30q20A	3527	4795.10	4917.30	20.61	3600.00	38196	1	5151	5151.00	4.75	1.05
n30q20B	3494	4753.92	4873.72	25.72	3600.00	60550	0	5336	5336.00	9.49	1.03
n30q20C	3605	4663.53	4799.08	16.10	3600.00	56650	0	5283	5283.00	10.08	1.34
n30q20D	3684	4701.90	4901.72	24.79	3600.00	45200	1	5136	5136.00	4.78	1.07
n30q20E	3559	4718.71	4877.90	22.18	3600.00	51607	0	5558	5558.00	13.94	1.09
n30q20F	3287	4561.58	4778.21	14.07	3600.00	42990	0	5252	5252.00	9.92	1.19
n30q20G	4340	5660.45	5752.77	36.41	3600.00	42384	0	6821	6821.00	18.57	1.77
n30q20H	2971	4287.60	4404.00	17.53	115.18	736	0	**4404**	4404.00	0.00	0.99
n30q20I	3262	4392.19	4502.50	22.70	3600.00	40997	0	4868	4868.00	8.12	1.02
n30q20J	3572	4705.40	4779.97	28.65	3600.00	34706	0	5603	5603.00	17.22	1.34

Table 5: Summary of computational results for $n = 20$, $K = 3$ and $N = 2$

Instance	L	BC						ILS			
		Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n20q10A	2128	4527.84	4694.71	11.49	3600.00	141123	0	5226	5226.00	11.32	0.51
n20q10B	2325	4841.79	5090.87	6.26	3600.00	285603	0	5920	5920.00	16.29	0.56
n20q10C	2756	5790.44	5930.73	19.03	3600.00	70777	0	6777	6777.00	14.27	0.87
n20q10D	2695	5833.02	5974.18	16.59	3600.00	111050	0	6621	6621.00	10.83	0.86
n20q10E	2735	5917.96	6062.60	13.61	3600.00	107165	3	6785	6785.00	11.92	0.79
n20q10F	2466	4775.84	4965.30	9.30	3600.00	145776	0	5788	5788.00	16.57	0.91
n20q10G	2464	4967.24	5217.98	8.13	3600.00	150975	3	5893	5893.00	12.94	0.74
n20q10H	2824	5160.81	5530.86	8.68	3600.00	144000	0	7007	7007.00	26.69	0.72

Continued on the next page

Table 5: Results for $n = 20$, $K = 3$ and $N = 2$ (continued)

Instance	L	Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n20q10I	2360	4443.55	4653.54	11.58	3600.00	179762	8	5392	5392.00	15.87	0.69
n20q10J	2195	4335.67	4629.00	6.02	3600.00	225360	22	5402	5402.00	16.70	0.55
n20q15A	2128	3958.08	4226.79	6.65	3600.00	217378	0	5035	5035.00	19.12	0.48
n20q15B	2228	4593.24	4894.20	6.16	3600.00	280372	0	5231	5231.00	6.88	0.54
n20q15C	2520	4967.13	5178.01	11.85	3600.00	102448	0	6065	6065.00	17.13	0.69
n20q15D	2595	4862.84	5042.90	12.41	3600.00	118331	0	6258	6258.00	24.10	1.08
n20q15E	2517	5126.98	5403.39	11.49	3600.00	146582	0	6193	6193.00	14.61	0.51
n20q15F	2427	4631.84	4801.06	8.91	3600.00	137572	0	5660	5660.00	17.89	0.96
n20q15G	2258	4855.87	5271.43	6.16	3600.00	148142	0	5717	5717.00	8.45	0.51
n20q15H	2824	4716.51	5185.59	7.06	3600.00	169200	6	6629	6629.00	27.84	0.65
n20q15I	2146	4265.40	4517.09	6.99	3600.00	162444	1	5028	5028.00	11.31	0.57
n20q15J	2195	4050.99	4407.47	5.24	3600.00	228356	5	5300	5300.00	20.25	0.56
n20q20A	2128	3838.09	4140.81	3.34	3600.00	287030	0	5035	5035.00	21.59	0.42
n20q20B	2228	4420.11	4837.81	4.29	3600.00	373834	9	5231	5231.00	8.13	0.61
n20q20C	2469	4572.67	4836.59	8.96	3600.00	101872	0	5926	5926.00	22.52	0.75
n20q20D	2396	4463.11	4747.07	9.21	3600.00	137628	0	5674	5674.00	19.53	0.85
n20q20E	2407	4827.13	5048.87	8.36	3600.00	165175	0	5988	5988.00	18.60	0.75
n20q20F	2450	4558.05	4789.32	6.12	3600.00	169943	0	5404	5404.00	12.83	0.65
n20q20G	2258	4774.02	5148.33	5.46	3600.00	181988	0	5717	5717.00	11.05	0.60
n20q20H	2598	4536.83	5246.77	5.48	3600.00	183102	0	6166	6166.00	17.52	0.63
n20q20I	2146	4162.01	4471.14	5.66	3600.00	194176	0	4954	4954.00	10.80	0.56
n20q20J	2195	3971.67	4370.69	5.37	3600.00	226184	11	5278	5278.00	20.76	0.45

Table 6: Summary of computational results for $n = 30$, $K = 3$ and $N = 2$

		BC						ILS			
Instance	L	Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n30q10A	2874	5934.96	5982.56	51.37	3600.00	30318	0	6875	6876.10	14.94	2.10
n30q10B	2859	6043.92	6185.79	27.40	3600.00	37954	0	6972	6972.00	12.71	1.48
n30q10C	3090	5778.69	5868.04	33.22	3600.00	48706	0	7108	7116.90	21.28	2.38
n30q10D	3015	5542.38	5653.90	25.27	3600.00	41034	1	6525	6525.80	15.42	1.88
n30q10E	2991	5617.27	5694.02	37.17	3600.00	40486	0	7122	7122.00	25.08	1.53
n30q10F	2789	5389.12	5506.16	33.79	3600.00	44332	0	6506	6506.00	18.16	1.45
n30q10G	3858	8174.72	8264.69	38.06	3600.00	39822	0	9434	9434.00	14.15	1.85
n30q10H	2919	5536.95	5649.05	44.58	3600.00	37030	0	6698	6804.10	20.45	1.66
n30q10I	2719	5053.92	5149.49	41.14	3600.00	40347	0	6056	6056.00	17.60	1.45
n30q10J	3003	5657.10	5711.10	41.77	3600.00	33747	0	6538	6538.00	14.48	2.12
n30q15A	2743	5191.35	5322.57	34.51	3600.00	30155	0	6243	6243.00	17.29	1.37
n30q15B	2647	5231.57	5318.63	28.43	3600.00	48608	0	6110	6110.00	14.88	1.24
n30q15C	2747	5176.66	5303.31	25.39	3600.00	51881	0	6236	6236.00	17.59	2.88
n30q15D	2680	5106.61	5256.13	23.07	3600.00	51297	0	5798	5798.00	10.31	1.58
n30q15E	2785	5100.24	5219.16	20.17	3600.00	52500	0	6337	6337.00	21.42	1.69
n30q15F	2490	4980.90	5143.51	19.47	3600.00	49170	0	5915	5915.00	15.00	1.21
n30q15G	3255	6638.72	6720.82	41.69	3600.00	36531	0	7600	7600.00	13.08	1.74

Continued on the next page

Table 6: Results for $n = 30$, $K = 3$ and $N = 2$ (continued)

Instance	L	Root LB	LB	Root Time (s)	Time (s)	Tree Size	#Lazy Cuts	Min Cost	Avg Cost	Avg Gap (%)	Avg Time (s)
n30q15H	2606	4747.45	4813.97	41.93	3600.00	43841	0	6102	6102.00	26.76	1.80
n30q15I	2501	4639.79	4727.24	24.81	3600.00	41046	0	5526	5526.00	16.90	1.29
n30q15J	2796	5095.13	5193.83	29.05	3600.00	42921	0	6034	6034.00	16.18	1.42
n30q20A	2482	5090.96	5223.36	20.96	3600.00	39732	0	5769	5769.00	10.45	1.01
n30q20B	2647	4960.88	5199.03	26.26	3600.00	45534	0	6086	6086.00	17.06	1.29
n30q20C	2632	4920.93	5034.62	16.14	3600.00	64445	0	5731	5731.00	13.83	1.29
n30q20D	2568	5009.92	5154.88	21.52	3600.00	55724	0	5872	5872.00	13.91	1.18
n30q20E	2785	4963.61	5088.05	19.38	3600.00	52594	0	6014	6014.00	18.20	1.34
n30q20F	2490	4874.49	5082.34	15.04	3600.00	62524	0	5601	5601.00	10.21	0.92
n30q20G	3014	5964.44	6058.46	40.35	3600.00	40978	0	6968	6968.00	15.01	0.94
n30q20H	2502	4466.08	4626.66	18.02	3600.00	63658	0	5679	5679.00	22.75	1.90
n30q20I	2393	4599.53	4737.05	20.08	3600.00	46715	0	5410	5410.00	14.21	1.07
n30q20J	2692	4962.80	5037.11	33.22	3600.00	44192	0	5980	5980.00	18.72	1.09

For almost all cases, BC could not finish its execution within the time limit of one hour. Yet, the exact algorithm managed to solve seven instances to optimality, where six of them are for $n = 20$ and $K = 2$, and the other one is for $n = 30$ and $K = 2$. Note that ILS also found the optimal solutions for these seven instances, but in less than one second. BC could not prove the optimality of any of the instances with $K = 3$. It also failed to find an integer solution for many instances, which possibly explains the lack of lazy cuts in such cases, especially for $K = 3$. As expected, the root time for $n = 30$ is larger than $n = 20$, because the CPU time required to solve the linear programs naturally increases with the size of the instance. Since this also happens in the other nodes, a smaller number of nodes could be solved within the time limit for the instances involving 30 stations, when compared to those containing 20 stations. ILS found always the same solution for the 10 runs in all cases, except for instances n30q10H, n30q15G, n30q10A, n30q10C, n30q10H, thus suggesting that the proposed heuristic has a consistent performance in terms of robustness. The algorithm also runs very fast for these instances, never spending, on average, more than three seconds (except for instance n30q10C).

5.5. Aggregate results

Tables 7 and 8 show the aggregate results for $K = 2$ and $K = 3$, respectively, where $\mathbf{Gap_{LB}}$ denotes the average gap between the average solutions and the lower bound found by BC, $\mathbf{Gap_{BKS}}$ represents the average gap between the average solutions and the best known solutions (BKSs) and **Time (s)** corresponds to the average CPU time in seconds. We do not report the gaps with respect to the lower bound for the instances involving 200 stations, because BC failed in most cases to solve the linear relaxation within the time limit.

Table 7: Aggregate results for $K = 2$

n	$Q = 10$			$Q = 15$			$Q = 20$		
	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)
20	10.09	0.00	0.72	8.43	0.00	0.66	6.42	0.00	0.57
30	13.57	0.06	2.16	13.65	0.03	1.71	9.69	0.00	1.19
40	13.84	0.15	3.69	14.58	0.01	2.79	12.75	0.01	2.61
50	14.67	0.56	7.14	12.20	0.06	5.79	10.46	0.13	4.10
60	17.47	1.32	11.29	14.37	0.55	6.92	11.91	0.17	7.05
100	20.87	2.11	36.10	18.65	1.72	37.25	15.11	0.92	27.39
200	-	1.79	255.62	-	1.87	218.62	-	1.93	262.71

Table 8: Aggregate results for $K = 3$

n	$Q = 10$			$Q = 15$			$Q = 20$		
	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)	Gap_{LB} (%)	Gap_{BKS} (%)	Time (s)
20	15.34	0.00	0.72	16.76	0.00	0.66	16.33	0.00	0.63
30	17.43	0.17	1.79	16.94	0.00	1.62	15.43	0.00	1.20
40	18.05	0.41	3.41	17.06	0.14	2.59	16.31	0.00	2.43
50	16.62	0.89	6.19	14.72	0.17	4.80	13.70	0.12	4.45
60	19.63	1.43	10.25	15.98	0.38	8.02	15.08	0.21	7.54
100	22.02	2.30	37.99	20.24	2.03	34.40	17.02	1.40	27.10
200	-	2.00	268.19	-	2.00	225.05	-	2.35	233.31

The results demonstrate that there is a considerable gap between the average solution values and the lower bound. This does not necessarily mean that the solutions generated by ILS are of poor quality. In fact, based on the small values of the average gaps with respect to the BKSs, we suspect that the upper bounds found by ILS is likely to be closer to the optimal solution than the lower bounds obtained by BC. For example, for the instances involving up to 50 stations, it can be observed that the average gaps with respect to the BKSs are always smaller than 1%, which suggests that ILS systematically finds, on average, potentially high quality solutions. Furthermore, the average CPU time of the proposed heuristic can be considered acceptable, especially for the instances containing up to 100 stations.

It is important to point ot that we have disregarded the runs where ILS was not capable of generating a feasible solution when computing the average gaps. Tables 9 and 10 illustrate those instances for $K = 2$ and $K = 3$, respectively, where the proposed heuristic failed to find a feasible solution in at least one of the 10 runs.

There are relatively very few instances where ILS was not successful in finding 10 feasible solutions out of 10 runs. This happened in 17 instances for $K = 2$ (8.09%) and in 25 instances for $K = 3$ (11.09%). We can also observe that, in most cases, such instances

Table 9: Instances with less than 10 feasible runs for $K = 2$

Number of feasible runs and corresponding instances								
1	2	3	4	5	6	7	8	9
n200q10F	n200q10D	-	n200q10C	n200q10H	n50q20D	n100q10B	n100q10A	n100q15E
					n200q15E	n200q10J	n200q15C	n100q10D
					n200q15F	n200q10B	n200q15A	
					n200q10A			
					n200q20E			

Table 10: Instances with less than 10 feasible runs for $K = 3$

Number of feasible runs and corresponding instances								
1	2	3	4	5	6	7	8	9
n200q15D	n200q10J	-	n200q15F	-	n200q15A	n100q10D	n60q20I	n40q10C
	n200q10F		n200q10B			n200q15B	n100q20C	n40q15H
			n200q10H			n200q15H	n200q10A	n40q15I
			n200q10C				n200q10E	n50q20I
							n200q20E	n100q10I
								n100q15B
								n100q15H
								n100q10A
								n200q20F

contain 200 stations, especially for $K = 2$. The results suggests that, depending on the value of L, finding feasible SBRP-MVV solutions in a systematic fashion is a challenging task.

6. Concluding remarks

We have presented the *static bicycle relocation problem with multiple vehicles and visits* (SBRP-MVV), and we have proposed a branch-and-cut (BC) algorithm over an extended network-based mathematical formulation. The constraints that ensure that a station is never visited by more than one vehicle are added in a lazy fashion, that is, only when an infeasible integer solution is found. In addition, we have also developed an iterated local search (ILS) based heuristic that uses efficient auxiliary data structures to perform move evaluations in amortized constant time during the local search. This is done by storing several information of each subsequence of a current solution.

Extensive computational experiments were conducted on new 1260 benchmark instances, ranging from to 20 to 200 stations. The results obtained revealed that multiple visits are only interesting for those instances whose vehicle capacity is up to 20. Moreover, the average gaps between the average solutions found by ILS and the lower bound obtained by BC appear to increase with the number of vehicles. On the other hand, the number of multiple visits is likely to decrease as the number of vehicles increases.

Acknowledgments

This work was partially supported by the Brazilian research agency CNPq, grant 305223/2015-1, and by the Canadian Natural Sciences and Engineering Research Council under grant 2015-06189. This support is gratefully acknowledged.

References

G. Laporte, F. Meunier, R. Wolfler Calvo, Shared mobility systems, 4OR 13 (4) (2015) 341–360.

R. Nair, E. Miller-Hooks, Fleet Management for Vehicle Sharing Operations, Transportation Science 45 (2011) 524–540.

C. Contardo, C. Morency, L.-M. Rousseau, Balancing a dynamic public bike-sharing system, Tech. Rep., CIRRELT-2012-09, Montréal, Canada, 2012.

D. Chemla, F. Meunier, T. Pradeau, R. Wolfler Calvo, H. Yahiaoui, Self-service bike sharing systems: simulation, repositioning, pricing, Tech. Rep. hal-00824078, Centre d'Enseignement et de Recherche en Mathématiques et Calcul Scientifique - CERMICS, Laboratoire d'Informatique de Paris-Nord - LIPN , Parallélisme, Réseaux, Systèmes d'information, Modélisation - PRISM, 2013a.

M. Benchimol, P. Benchimol, B. Chappert, A. De La Taille, F. Laroche, F. Meunier, L. Robinet, Balancing the stations of a self-service bike hire system, RAIRO-Operations Research 45 (2011) 37–61.

T. Raviv, M. Tzur, I. Forma, Static repositioning in a bike-sharing system: models and solution approaches, EURO Journal on Transportation and Logistics 2 (2013) 187–229.

G. Erdoğan, G. Laporte, R. Wolfler Calvo, The static bicycle relocation problem with demand intervals, European Journal of Operational Research 238 (2014) 451–457.

Y. Li, W. Y. Szeto, J. Long, C. S. Shui, A multiple type bike repositioning problem, Transportation Research Part B: Methodological 90 (2016) 263–278, ISSN 0191-2615.

J. H. Lin, T. C. Chou, A Geo-Aware and VRP-Based Public Bicycle Redistribution System, International Journal of Vehicular Technology 2012 (2012) 1–14.

S. C. Ho, W. Y. Szeto, Solving a static repositioning problem in bike-sharing systems using iterated tabu search, Transportation Research Part E: Logistics and Transportation Review 69 (2014) 180–198, ISSN 1366-5545.

M. Dell'Amico, E. Hadjicostantinou, M. Iori, S. Novellani, The bike sharing rebalancing problem: Mathematical formulations and benchmark instances, Omega 45 (2014) 7–19.

I. A. Forma, T. Raviv, M. Tzur, A 3-step math heuristic for the static repositioning problem in bike-sharing systems, Transportation Research Part B: Methodological 71 (2015) 230–247, ISSN 0191-2615.

M. Dell'Amico, M. Iori, S. Novellani, T. Stützle, A destroy and repair algorithm for the Bike sharing Rebalancing Problem, Computers & Operations Research 71 (2016) 149–162.

G. Erdoğan, M. Battarra, R. Wolfler Calvo, An exact algorithm for the static rebalancing problem arising in bicycle sharing systems, European Journal of Operational Research 245 (3) (2015) 667–679, ISSN 0377-2217.

L. D. Gaspero, A. Rendl, T. Urli, A Hybrid ACO+CP for Balancing Bicycle Sharing Systems, in: M. Blesa, C. Blum, P. Festa, A. Roli, M. Sampels (Eds.), Hybrid Metaheuristics, vol. 7919 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, 198–212, 2013.

M. Rainer-Harbach, P. Papazek, G. Raidl, B. Hu, C. Kloimullner, PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems, Journal of Global Optimization (2014) 1–33.

D. Chemla, F. Meunier, R. Wolfler Calvo, Bike sharing systems: Solving the static rebalancing problem, Discrete Optimization 10 (2013b) 120–146.

R. Alvarez-Valdes, J. M. Belenguer, E. Benavent, J. D. Bermudez, F. Muñoz, E. Vercher, F. Verdejo, Optimizing the level of service quality of a bike-sharing system, Omega 62 (2016) 163–175.

F. Cruz, A. Subramanian, B. P. Bruck, M. Iori, A heuristic algorithm for a single vehicle static bike sharing rebalancing problem, Submitted to Computers & Operations Research URL http://arxiv.org/abs/1605.00702.

H. R. Lourenço, O. C. Martin, T. Stützle, Iterated Local Search: Framework and Applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, vol. 146

of *International Series in Operations Research & Management Science*, Springer, New York, 363–397, 2010.

A. Subramanian, L. M. A. Drummond, C. Bentes, L. S. Ochi, R. Farias, A parallel heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery, Computers & Operations Research 37 (11) (2010) 1899–1911.

P. H. V. Penna, A. Subramanian, L. S. Ochi, An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem, Journal of Heuristics 19 (2013) 201–232.

T. Vidal, M. Battarra, A. Subramanian, G. Erdoğan, Hybrid Metaheuristics for the Clustered Vehicle Routing Problem, Computers & Operations Research. 58 (2015) 87–99, ISSN 0305-0548.

M. M. Silva, A. Subramanian, L. S. Ochi, An iterated local search heuristic for the split delivery vehicle routing problem, Computers & Operations Research 53 (0) (2015) 234–249, ISSN 0305-0548.

H. Hernández-Pérez, J.-J. Salazar-González, Heuristics for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem, Transportation Science 38 (2) (2004) 245–255.

M. G. Resende, C. C. Ribeiro, Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, vol. 146 of *International Series in Operations Research & Management Science*, Springer, New York, 283–319, 2010.

A. Subramanian, Heuristic, Exact and Hybrid Approaches for Vehicle Routing Problems, Ph.D. thesis, Programa de Pós-Graduação em Computação, Universidade Federal Fluminense, 2012.

N. Mladenović, P. Hansen, Variable Neighborhood Search, Computers & Operations Research 24 (1997) 1097–1100.