Annals of Operations Research (2022) 316:979–1011
https://doi.org/10.1007/s10479-020-03789-0

S.I.: CLAIO 2018

A multi-start evolutionary local search for the one-commodity pickup and delivery traveling salesman problem

Juan D. Palacio[1] · **Juan Carlos Rivera**[1]

Accepted: 9 September 2020 / Published online: 20 September 2020

Abstract
This article addresses the one-commodity pickup and delivery traveling salesman problem (1-PDTSP), which is a generalization of the well-known traveling salesman problem. The 1-PDTSP aims to find a Hamiltonian tour in which a set of supply points (pickup locations), demand points (delivery locations) are visited and, the total traveled distance is minimized. We propose a hybrid metaheuristic based on multi-start evolutionary local search and variable neighborhood descent to solve the 1-PDTSP. To test the performance of our algorithm, we solve instances with up to 500 nodes available in the literature and we demonstrate that our approach is able to provide competitive results when comparing to other existing strategies. Since a direct application of the 1-PDTSP arises as the bicycle repositioning problem, we also use our metaheuristic algorithm to solve a set of real-case instances based on EnCicla, the bicycle sharing system in the Aburrá Valley (Antioquia, Colombia).

Keywords Pickup and delivery traveling salesman problem · Evolutionary local search · Variable neighborhood descent · Bicycle repositioning problem

1 Introduction

In the one-commodity pickup and delivery traveling salesman problem (1-PDTSP) a set of locations and a capacitated vehicle are given. The locations are classified as supply points (pickup locations) and demand points (delivery locations). One commodity is transported between the locations using the available vehicle. Each supply point provides a certain amount of the commodity and these units can be delivered to one or several demand points (Hernández-Pérez and Salazar-González 2004b). Additionally, one of these locations is set as a depot where the vehicle starts and ends the tour. Thus, the 1-PDTSP aims to find a

✉ Juan D. Palacio
jpalac26@eafit.edu.co

Juan Carlos Rivera
jrivera6@eafit.edu.co

[1] Mathematical Modeling Research Group. Mathematical Sciences Department, School of Sciences, Universidad EAFIT, Medellín, Colombia

Hamiltonian tour in which the total traveled distance is minimized and the vehicle capacity is satisfied. Moreover, an initial load must be determined for the vehicle.

A well-known application for the 1-PDTSP relies on the operation of bicycle sharing systems (BSSs). A BSS is mainly composed by a set of stations with limited capacity (i.e., number of bike slots) distributed along an urban area and also, a set of available bicycles for users. Citizens can use the system by taking a bike from one of the available stations (origin) and then, after a short trip, returning it to the same or a different station (destination). A successful operation of a BSS requires an adequate number of available bikes and parking slots at each station in such a way that the demand is satisfied and the station capacity is not violated. To do so, at least one capacitated vehicle must visit the stations in order to pick up or deliver bikes according to the lack or surplus at each location. From an optimization perspective, the design of routes to ensure the system demand requirements is well known as a bicycle repositioning problem (BRP) in BSS contexts. The BRP has been mainly tackled in its static version which assumes that there are no changes on stations demands (this mainly occurs at night when BSSs are not available for users). On the other hand, on dynamic BRPs, pickup and delivery quantities may vary (e.g. the repositioning operation is performed during the BSS operation).

So far, several mathematical programming models and exact solution strategies have been developed for the 1-PDTSP or the BRP. Nevertheless, given the $\mathcal{NP}$-hardness nature of the problems (Hernández-Pérez and Salazar-González 2004a), these strategies are not suitable to deal with large instances. Then, to solve the 1-PDTSP, we propose a hybrid metaheuristic which combines an evolutionary local search (ELS) with a variable neighborhood descent (VND) algorithm. Although Hernández-Pérez et al. (2009) describe a hybrid algorithm based on GRASP and VND for the 1-PDTSP, to the best of our knowledge, there is not report of hybrid ELS algorithms as solution strategies for the 1-PDTSP or the static BRP. In this paper, we explore the advantages of ELS over GRASP for the 1-PDTSP since the evolutionary algorithm better sample solutions near to a local optimum before leaving it (Duhamel et al. 2011). Moreover, apart from that evolutionary nature of our algorithm and its inherent perturbation procedure, we also state the following contributions for our work that are not considered in Hernández-Pérez et al. (2009): firstly, we include a family of well-known neighborhoods for TSP based on $Or - opt(\lambda)$ (Or 1976) in our VND. Secondly, our ELS does not deal with repairing procedures when unfeasible solutions appear, we discard them during the construction phase of the algorithm. Thus, our hybrid approach searches in a tighter solution space and we do not require the use of any repairing procedure. Lastly, our solution approach is able to find better solutions that the ones reported in Hernández-Pérez et al. (2009). This occurs even when we set a smaller size for the VND neighborhoods to improve 1-PDTSP solutions. Finally, based on our computational experiments, we show the benefits of the evolutionary local search metaheuristic within a multi-start procedure as GRASP. This is also a key factor to differentiate the GRASP algorithm for the 1-PDTSP in Hernández-Pérez et al. (2009) and our MS-ELS.

This paper is organized as follows. Firstly, we provide a brief description of background about models and solution strategies for the 1-PDTSP and its known application, the BRP in Sect. 2. Then, Sect. 3 presents a formal problem description for the 1-PDTSP while in Sect. 4 the proposed metaheuristic strategy and its variants are described. Next, in Sect. 5, we report the main results of our experiments using public instances for the 1-PDTSP and instances related to a BSS in Aburrá Valley (Antioquia, Colombia). Finally, Sect. 6 summarizes the main conclusions and points out some research directions.

2 Literature review

We devote this section to briefly summarize the related work on 1-PDTSP and BRPs. We firstly describe reported models and solution strategies for the 1-PDTSP and then, we show some of the applications based on BSS contexts.

Initially, Hernández-Pérez and Salazar-González (2004a) introduce the 1-PDTSP. They also develop a branch-and-cut (B&C) algorithm based on an integer linear programming (ILP) model in which the total travel distance is minimized. The solution strategy is based on Benders cuts, Clique inequalities generation and also, an adaptation of *nearest insertion* as a heuristic to strengthen upper bounds in the tree. Optimal solutions are reported for instances with up to 50 nodes. Later, in Hernández-Pérez and Salazar-González (2004b), authors propose two different heuristics. The first one, based on *nearest neighbor*, compute new values in the distance matrix in order to penalize movements leading to unfeasible solutions (e.g., by avoiding edges connecting nodes with similar demand). The second heuristic consists on modifying the original B&C algorithm to include inequalities is such a way a subset of neighbor solutions can be explored after an initial solution for the 1-PDTSP is found. Both heuristic strategies are able to find small gaps even when values for the vehicle capacity is tight and the number of nodes is not larger than 60.

Hernández-Pérez et al. (2009) implement a hybrid GRASP metaheuristic in which a VND is used in the improvement phase, minimizing the tour total distance for the 1-PDTSP. After the constructive phase, the local search is replaced by a VND based on edge exchange neighborhoods (2-*opt* and 3-*opt*). After the GRASP scheme is executed, a second VND performs a post-optimization phase using vertex exchange neighborhoods (*forward* and *backward* operators). The instances are randomly generated and the number of nodes vary from 20 to 500 while the vehicle capacity ranges between 10 and 40. In the latter case, when the capacity is large enough, the solution of the 1-PDTSP is equal to the one found by the classical TSP model (without pickups and deliveries feature). The GRASP/VND approach is able to find an optimal solution of 96.7% of the instances with up to 50 nodes while BKS are reported for larger instances. The authors compare their results with those reported in Hernández-Pérez and Salazar-González (2004a) and Hernández-Pérez and Salazar-González (2004b) for small and large instances, respectively. Zhao et al. (2009) describe a genetic algorithm (GA) with a pheromone-based crossover operator. This operator uses local and global information to generate new offspring. While the local information includes edge lengths, adjacency relations, and demands on nodes, the global information is based on pheromone trails. Each offspring is locally improved using a 2-*opt* and then, the mutation process is based on a simple 3−exchange operator. This evolutionary algorithm outperforms the hybrid GRASP/VND presented in Hernández-Pérez et al. (2009) for instances up to 500 nodes, and many new best known solutions for large instances are reported. Later, Mladenović et al. (2012) describe a variable neighborhood search procedure with four neighborhoods based on double-bridge and insertion operators. This strategy is able to solve 1-PDTSP instances with up to 1000 nodes.

As mentioned in Sect. 1, one of the most known applications for the 1-PDTSP arises in BSS logistic operations and it is called the static BRP. For details on some dynamic models and their solution strategies, we refer the reader to Contardo et al. (2012), Caggiani and Ottomanelli (2013), Kloimüllner et al. (2014), Zhang et al. (2017), Shui and Szeto (2018) and Legros (2019). From a static perspective of the problem, Raviv et al. (2013) formulate two mixed-integer linear programming models (MILPs) for a BRP in which operational cost of the routes and the number of bike shortages in each station are minimized in a weighted

objective. The MILPs are compared by solving instances with up to 60 stations based on certain locations of Vélib (BSS in Paris) and then, a real instance with 104 stations tested with one or two vehicles. Chemla et al. (2013) also propose a B&C procedure for the BRP. This algorithm is based on a MILP relaxation for the problem and provides lower bounds when several visits to each node are allowed. A tabu search (TS) with four different neighborhoods is also designed to find upper bounds. For instances with up to 60 stations average gaps over 5% are reported. For the set of largest instances, gaps increase and the local search procedure is not quite efficient since the size of neighborhoods grows significantly.

Dell'Amico et al. (2014) present four MILPs for the multi-vehicle BRP. They solve these mathematical models via B&C algorithm. The authors compare the performance of the proposed formulations and confirm that the problem complexity not only relies on the number of nodes but also on vehicle capacity and maximum demand on stations. The solution strategy was tested in 65 instances adapted from 22 different BSSs around the world and the formulation with best computational performance is able to solve optimally all the instances with up to 50 nodes in less than 15 min. Similar to Raviv et al. (2013) and Ho and Szeto (2014) model penalty functions to minimize the cost associated to unsatisfied demand in a BSS. To deal with the single-vehicle case, the authors use an iterated TS procedure and also solve a MILP using CPLEX. The TS includes three different neighborhoods based on removal, insertion and exchange moves and the computational experiments are based on instances up to 400 stations.

Forma et al. (2015) propose a 3-step matheuristic based on a clustering process supported on savings heuristic and two MILPs for routing and repositioning in each cluster. The authors also use a weighted sum to minimize penalization for bike shortages and the total distance or operational cost of the route. For instances up to 150 stations, the matheuristic outperforms the formulations in Raviv et al. (2013) obtaining smaller gaps. Dell'Amico et al. (2016) propose a destroy and repair (D&R) metaheuristic for the BRP and for the one-commodity pickup and delivery vehicle routing problem with maximum duration (1-PDVRPD) in which a maximum route length constraint is added to the BRP. This D&R metaheuristic is tested on instances with up to 500 stations. For less than 50 stations, it is possible to find optimal solutions and best known solutions for some larger instances are presented. Szeto et al. (2016) design a variant of chemical reaction optimization (CRO) procedure called enhanced CRO. They address the single-vehicle case in order to minimize a weighted sum of the unmet customer demand and the total time of the route. By testing instances up to 300 stations, it is possible to conclude that the enhanced CRO outperforms not only the classical version of the algorithm but also a proposed MILP solved in CPLEX. Ho and Szeto (2017) propose a hybrid large neighborhood search (HLNS) algorithm to deal with penalty function for unsatisfied demands as the goal of the BRP. This hybrid metaheuristic includes five removal (destroy) operators, five insertion (repair) operators and a TS applied to the most promising solutions. Testing instances with up to 518 stations, the HLNS is able to outperform a proposed MILP solved by CPLEX and the matheuristic described in Forma et al. (2015).

3 The one-commodity pickup and delivery traveling salesman problem

In this section, we formally describe the one-commodity pickup and delivery traveling salesman problem (1-PDTSP). The 1-PDTSP is modeled on a complete graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{0, 1, \ldots, n\}$ is the set of nodes and $\mathcal{A}$ is the set of arcs between each pair of nodes. Without loss of generality, the location 0 is the depot but it is also considered as a node in

our problem. For each arc $(i, j) \in \mathcal{A}$, we define a positive traveling cost from i to j ($i \neq j$) as c_{ij}. Additionally, for each node i ($i \in \mathcal{N}$), we also state a parameter q_i (where $q_i \in \mathbb{Z}$) that represents the demand on node i. If $q_i < 0$, then node i requests a pickup of $|q_i|$ units. On the other hand, if $q_i > 0$, then a delivery operation is requested in node i and q_i units must be unloaded. These operations are performed by one available vehicle with capacity Q. With the aim to describe a mathematical model for the 1-PDTSP, we state a binary decision variable, y_{ij} where $(i, j) \in \mathcal{A}$. Variable y_{ij} takes the value of one if the vehicle traverses the arc (i, j) and zero otherwise. Additionally, a variable x_{ij} denotes the load of the vehicle if arc (i, j) is used in the solution. Finally, we also define variable z_{ij} that saves the order in which the arc (i, j) is used in the solution. Our mathematical formulation for the 1-PDTSP is a mixed-integer linear programming model (MILP) described by Eqs. (1)–(9) which was previously presented in Palacio and Rivera (2019) as follows:

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot y_{ij} \tag{1}$$

subject to,

$$\sum_{\substack{j\in\mathcal{N} \\ i\neq j}} y_{ij} = 1, \qquad \forall\, i \in \mathcal{N} \tag{2}$$

$$\sum_{j\in\mathcal{N}} y_{ij} = \sum_{j\in\mathcal{N}} y_{ji}, \qquad \forall\, i \in \mathcal{N} \tag{3}$$

$$x_{ij} \leq Q \cdot y_{ij}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{4}$$

$$\sum_{j\in\mathcal{N}} x_{ji} - \sum_{j\in\mathcal{N}} x_{ij} = q_i, \qquad \forall\, i \in \mathcal{N} \tag{5}$$

$$\sum_{j\in\mathcal{N}} z_{ji} - \sum_{j\in\mathcal{N}} z_{ij} = 1, \qquad \forall\, i \in \mathcal{N}\backslash\{0\} \tag{6}$$

$$z_{ij} \leq |\mathcal{N}| \cdot y_{ij}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{7}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{8}$$

$$z_{ij}, x_{ij} \geq 0, \qquad \forall\, (i, j) \in \mathcal{A} \tag{9}$$

The objective function in (1) aims to minimize the total traveled distance. Equations in (2) ensure that each node is visited exactly once, while constraints in (3) enforce to leave a node once it is visited. Constraints in (4) limit the maximum load when traversing any arc, to the vehicle capacity. Equation (5) force the model to ensure a flow conservation along the used arcs. Inspired on Miller et al. (1960), Eq. (6) state a coefficient for each arc and avoid subtours in the solution. Constraints in (7) limit arc coefficients. It is worth to mention that for classical TSP and VRP formulations, constraints (4) and (5) prevent subtours. Nevertheless, for pickup and delivery operations, as q_i can be negative, (4) and (5) are not enough. Constraints in (8) and (9) define the domain of decision variables. Note that variables x_{ij} and z_{ij} could be integer, but given the structure of our formulation, a continuous domain will lead to integer values.

Each node in $\mathcal{N}$, even the depot (node 0) is visited exactly once [see Eq. (2)]. However, we assume that the depot is able to absorb or provide the remaining number of units to ensure flow conservation for any 1-PDTSP instance as follows:

$$q_0 = -\sum_{i=1}^{|\mathcal{N}|} q_i \tag{10}$$

Conditions described in (2) also force the vehicle to visit even the nodes with demand equal to zero. Note that if nodes where $q_i = 0$ must not be visited, they can be removed from the set $\mathcal{N}$ as a preprocessing procedure. To describe graphically a 1-PDTSP solution, we present in Fig. 1, the optimal tour for a vehicle with capacity $Q = 10$ considering an instance with 20 nodes. While the first value near to each location represents the number of units to pick up or deliver, the second one denotes the load of the vehicle when entering to such location. To compute this load, we define l_k as the number of units in the vehicle after visiting k nodes. Thus, if i is the k-th visited location, l_k is computed as: $l_k = l_{k-1} - q_i$. For example, in this solution, the vehicle traverses the arc (0, 10) with a load of 10 units. Then, in node 10, nine units are delivered and the vehicle arrives to node 9 with a load equal to one. In node 9 there is not pickup or delivery operation, thus arc (9, 18) is traversed with one unit in the vehicle. In location 18, the vehicles picks up seven units. Note that at the end of the path, the vehicle goes into 0 with three units which represent the same initial load to begin the tour.

Note that any Hamiltonian tour is a feasible solution for the 1-PDTSP if the load of the vehicle does not exceed the capacity Q. Without loss of generality we may also say that a solution is feasible if the difference between the minimum and the maximum load of the vehicle is not greater than Q. This is a straightforward way to check feasibility in a 1-PDTSP solution:

$$\max_{i \in \mathcal{N}}\{l_i\} - \min_{i \in \mathcal{N}}\{l_i\} \leq Q \tag{11}$$

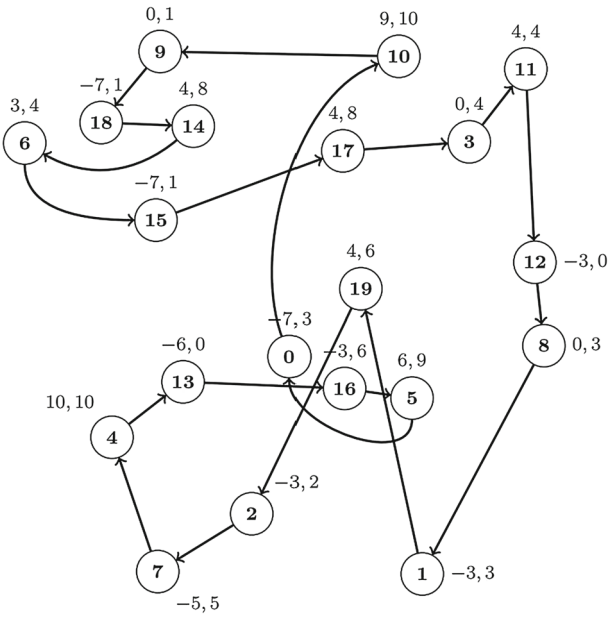

Fig. 1 Optimal solution for a 1-PDTSP instance with 20 nodes (Palacio and Rivera 2019)

4 Multi-start evolutionary local search for the 1-PDTSP

In this section, we present a multi-start evolutionary local search (MS-ELS) to deal with the 1-PDTSP. We firstly describe the general structure of the procedure. Then, we provide some details about the main components of the proposed algorithm: construction phase based on a greedy randomized procedure, improvement phase designed as a VND and the perturbation operator. Finally, we also briefly describe GRASP and ILS as two particular cases of MS-ELS. In spite of the good performance of this algorithms in related routing problems, evolutionary components within these strategies have shown also promising results (Prins 2009).

Evolutionary local search procedure is a metaheuristic framework mainly based on evolutionary strategies and local search algorithm. ELS is initially presented in Wolf and Merz (2007) for the solution of optimization problems in telecommunications. However, routing problems have been also addressed with ELS procedures. Some of these problems are the capacitated VRP (Prins 2009), the capacitated arc routing problem with split delivery (Belenguer et al. 2010), the truck and trailer routing problem (Villegas et al. 2010) and the multitrip cumulative capacitated vehicle routing problem (Rivera et al. 2013). In Fig. 2, we depict the main steps of a MS-ELS: after a starting solution is build and improved, perturbations are made for a number of iterations and best solutions update the incumbent in each iteration. This process continues until a number of start solutions are reached. MS-ELS can be also seen as a generalization of GRASP and multi-start iterated local search metaheuristics. Firstly, the greedy randomized construction heuristic ensures diversity in the search space. Then, the classical local search in GRASP, is replaced by an ELS which allows to explore in a better way the solution space near to a local optimum before restart a different search from a different starting solution (Duhamel et al. 2011).

4.1 MS-ELS framework for the 1-PDTSP

As mentioned before, ELS is a metaheuristic based on an evolutionary algorithm and a local search procedure. In an ELS, a single solution is constructed and then, improved via local search. For a number of iterations (`MaxIterations`), the obtained solution is perturbed and next, it is improved `MaxChildren` times. The algorithm returns the best overall solution found. In a multi-start ELS, the ELS is performed `MaxStarts` times. For the sake of clarity, hereinafter, we refer to a solution as a list of nodes. The order of the list denotes the path to follow by the vehicle, obtaining a hamiltonian cycle. Needless to say, for solution s, an objective function value is computed as sum of the travel costs traversing the arcs between nodes in the path.

The Algorithm 1 depicts our MS-ELS for the 1-PDTSP. Starting with an incumbent value for the objective function $\bar{z} = \infty$, `MaxStarts` solutions (s_0) are constructed via greedy randomized algorithm (line 4) and improved using a VND procedure (line 5). If the objective function of resulting solution s is better than the incumbent solution $\bar{s}$, the latter is updated (lines 6–9). For each one of the predefined `MaxIterations`, a set of `MaxChildren` perturbation processes are performed on s and immediately, the perturbed resulting solutions s_p are improved calling the VND procedure again (line 14). After each improvement, the algorithm updates the best solution on the current iteration (s') if improved (lines 15–18). Similarly, lines 20–23 update the best solution of the current start and lines 25–28 update the incumbent solution.

In the following sections, we describe the components of our algorithm depicted in Algorithm 1: Sect. 4.1.1 describes the construction procedure based on a greedy randomized

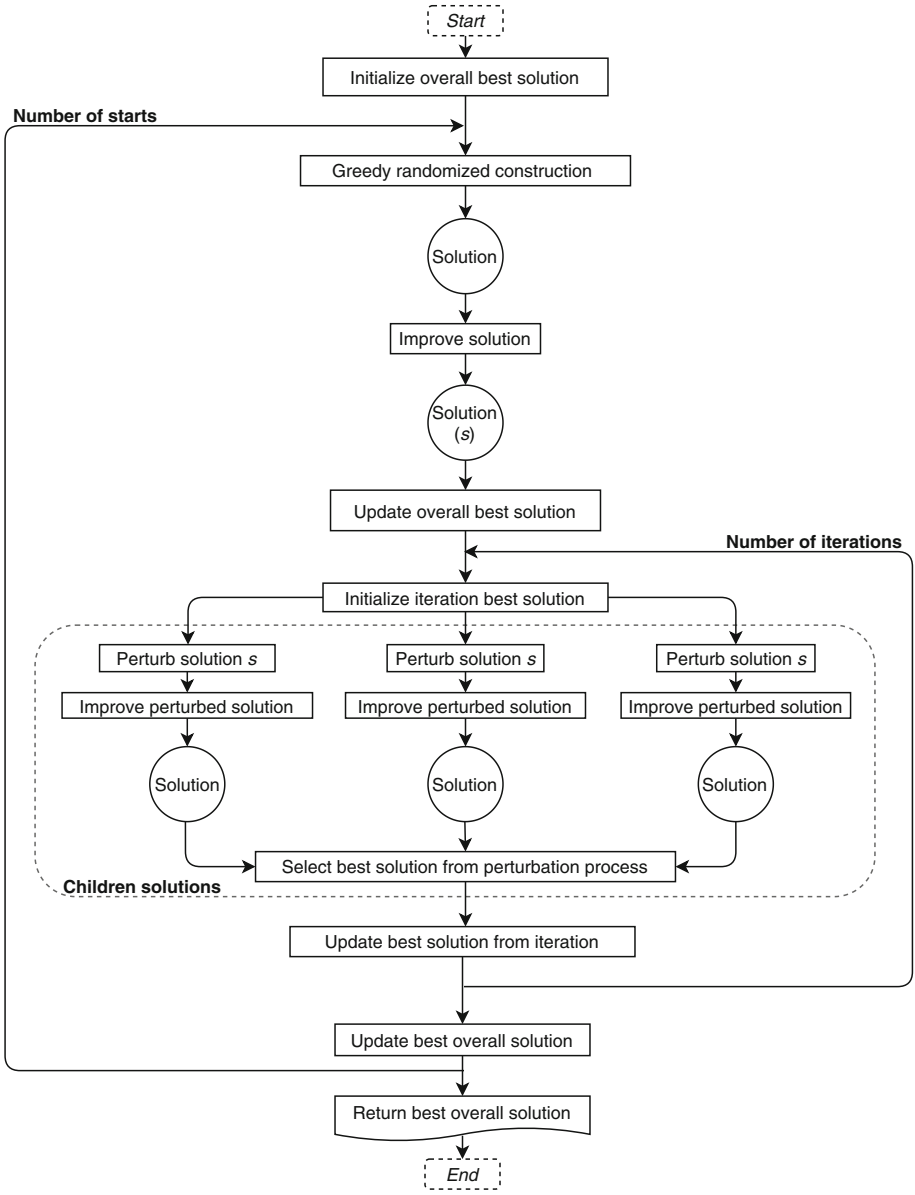

Fig. 2 Flow chart for MS-ELS algorithm

algorithm. Section 4.1.2 presents the structure and main components of the VND as improvement phase of the MS-ELS, and Sect. 4.1.3 briefly describes the perturbation procedure.

4.1.1 Greedy randomized construction

In order to generate several initial solutions, we use a greedy strategy based on the *nearest neighbor* algorithm. However, as Hernández-Pérez and Salazar-González (2004b) point out,

Algorithm 1 MS-ELS for the 1-PDTSP: general structure

```
1: function MS- ELS(MaxStarts, MaxIterations, MaxChildren)
2:    z̄ ⟵ ∞, s̄ ⟵ ∅
3:    for i = 1 to MaxStarts do
4:       s_0 ⟵ GreedyRandomizedAlgorithm(seed)
5:       s ⟵ VND(s_0)
6:       if f(s) < z̄ then
7:          s̄ ⟵ s
8:          z̄ ⟵ f(s)
9:       end if
10:      for j = 1 to MaxIterations do
11:         z′ ⟵ ∞, s′ ⟵ ∅
12:         for p = 1 to MaxChildren do
13:            s_p ⟵ Perturbation(s)
14:            s″ ⟵ VND(s_p)
15:            if f(s″) < z′ then
16:               s′ ⟵ s″
17:               z′ ⟵ f(s″)
18:            end if
19:         end for
20:         if f(s) < z′ then
21:            s′ ⟵ s
22:            z′ ⟵ f(s)
23:         end if
24:      end for
25:      if f(s′) < z̄ then
26:         s̄ ⟵ s′
27:         z̄ ⟵ f(s′)
28:      end if
29:   end for
30:   return s̄
31: end function
```

the nearest neighbor algorithm for the TSP, hardly finds a feasible 1-PDTSP solution if the vehicle capacity is tight. Then, following the procedure in Hernández-Pérez et al. (2009), we redefine the traveling costs, c_{ij}, for arcs $(i, j) \in \mathcal{A}$ by penalizing connections between pair of nodes of the same type (e.g. two nodes with pickup requests). Thus, new values for c_{ij} are stored in c'_{ij} $\forall$ $(i, j) \in \mathcal{A}$ as follows (Hernández-Pérez et al. 2009):

$$c'_{ij} = \begin{cases} c_{ij} + \frac{(K-Q)\cdot\sum_{(i,j)\in\mathcal{A}} c_{ij}}{10\cdot Q\cdot|\mathcal{N}|} \cdot (2Q - |q_i - q_j|) & \text{if } |q_i + q_j| \leq Q, \\ \infty & \text{otherwise,} \end{cases} \tag{12}$$

where K is the total sum of units delivered or picked up. Let us remark that as described in Eq. (10), 1-PDTSP instances are balanced in terms of the demands. Thus, the total units to deliver is equal to the total units to pick up. In order to illustrate our motivation to redefine traveling costs in c_{ij}, we present a feasible and an unfeasible solution for a 1-PDTSP instance with seven nodes in Figs. 3 and 4, respectively. For both solutions and similar to example in Fig. 1, the left side number near to each node, denotes its demand while the right side number represents the load of the vehicle when entering that node. Note that solution in Fig. 3 is a Hamiltonian tour in which a certain amount of units is delivered immediately after a pickup operation is performed. On the contrary, in Fig. 4, a constructive solution starts at depot (node 0) with an initial load of zero units. Then, nodes 2 and 6 are consecutively visited picking up seven and three units, respectively. Note that after leaving node 6 and delivering six units

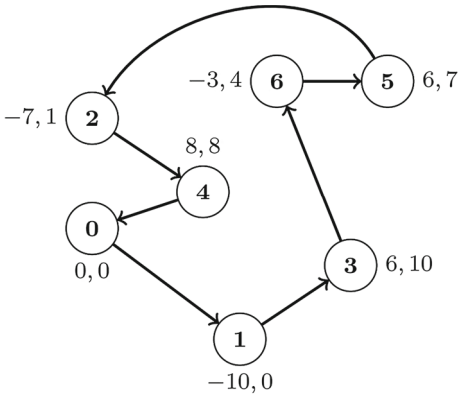

Fig. 3 Example of a feasible solution for the 1-PDTSP

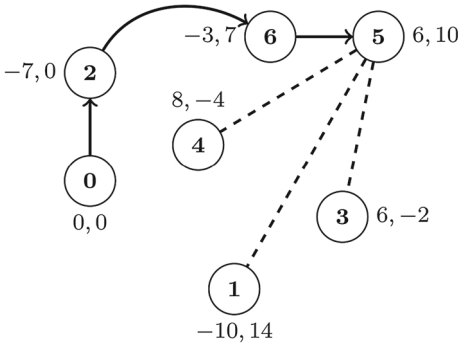

Fig. 4 Example of an unfeasible solution for the 1-PDTSP

at node 5, the solution become unfeasible since vehicle capacity is violated if any of the non-visited nodes (1, 3 and 4) are reached.

Once we compute values in c'_{ij}, the construction phase for a solution s follows:

(i) Select a node i as the first visited location at random. Let $p \longleftarrow 1$ and $s[p] \longleftarrow i$.
(ii) Define $\mathcal{R}$ as the set of closest and feasible (i.e. the vehicle capacity is not violated) non-visited nodes after the node i is served. Let $|\mathcal{R}|$ be the minimum between a restricted candidate list size (φ) and the number of feasible non-visited nodes. Note that if $\mathcal{R} = \emptyset$, the constructed solution so far leads to an unfeasible path as in Fig. 4.
(iii) If $|\mathcal{R}| > 0$, then choose a node j from $\mathcal{R}$ at random. If $\mathcal{R} = \emptyset$, then go to step (i).
(iv) Let $p \longleftarrow p + 1$ and $s[p] \longleftarrow j$. If $p < |\mathcal{N}|$ then define $i \longleftarrow j$ and go to step (ii), else stop.

Since each iteration of the constructive phase, requires to include a set of nodes in $\mathcal{R}$, we adapt the condition in (11), to check whether a location j is feasible to add in a partial solution with size $p - 1$ (Hernández-Pérez and Salazar-González 2004b):

$$\max_{i=1,\ldots,p-1}\{l_i, l_{p-1} - q_j\} - \min_{i=1,\ldots,p-1}\{l_i, l_{p-1} - q_j\} \leq Q \tag{13}$$

Let us comment that our proposed constructive strategy differs from the one presented in Hernández-Pérez et al. (2009). While our algorithm always end up with a feasible solution, the procedure described in Hernández-Pérez et al. (2009) does not guarantee a feasible result.

4.1.2 Variable neighborhood descent

VND is a deterministic variant of variable neighborhood search (VNS) which explores sequentially several neighborhoods represented by local search operators (Hansen and Mladenović 2001; Hansen et al. 2017). Given an incumbent solution s, the set of solutions reachable from s, $N_k(s)$, is explored when a local search operator k ($k \leq k_{max}$) is applied via function `LocalSearch`. If `LocalSearch` function retrieves a solution with smaller cost than $f(s)$, then s is updated and the search starts again from the first local search operator ($k = 1$). On the contrary, if s is not improved, then `LocalSearch` is performed using the operator $k+1$. The algorithm ends up if no improvement is found throughout the set of k_{max} neighborhoods.

As improvement phase for our metaheuristic algorithm, we propose the VND depicted in Algorithm 2. While the general structure of VND described above is mainly stated in lines 4–13 of the algorithm, we embed an additional function called `Reverse` (line 21) with the aim to explore different regions of the solution space. Function `Reverse`(s) simply changes the orientation of the path in solution s (i.e, the function returns s in the opposite direction). As mentioned in Hernández-Pérez and Salazar-González (2004b), the feasibility of solution s is independent of the orientation of the path. Then, since our strategy does not handle unfeasible solutions, function `Reverse` always delivers a feasible solution as well. Note that an additional cycle is added to our VND (line 3), just to ensure that after a solution is improved, function `Reverse` is applied. The sequential exploration based on local search operators is performed h times, with $h_{min} \leq h \leq h_{max}$, and each time, after first iteration, the operator `Reverse` is applied. This iterative procedure also may stop before, if no improvement is found after its first call ($b =$`true`).

Algorithm 2 Variable neighborhood descent for the 1-PDTSP

```
1: function VND(s, k_max, h_min, h_max)
2:     s' ⟵ s, h ⟵ 0, b ⟵False
3:     while (h ≤ h_max and b =False) or (h < h_min) do
4:         k ⟵ 1
5:         while k ≤ k_max do
6:             s'' ⟵ LocalSearch(s', N_k(s'))
7:             if f(s'') < f(s') then
8:                 s' ⟵ s''
9:                 k ⟵ 1
10:            else
11:                k ⟵ k + 1
12:            end if
13:        end while
14:        if f(s') < f(s) then
15:            s ⟵ s'
16:            b ⟵False
17:        else
18:            b ⟵True
19:        end if
20:        if h < h_max or b = False then
21:            s' ⟵ Reverse(s')
22:        end if
23:        h ⟵ h + 1
24:    end while
25:    return s'
26: end function
```

The sequential exploration based on several neighborhoods with the aim to improve solutions quality within our VND is based on the following two *edge-exchange* (EE) and five *chain-exchange* (CE) operators for the `LocalSearch` function. All operators follow the best improvement rule instead of first improvement criterion.

2-*opt and* 3-*opt* These neighborhoods are the first (for $k = 1$, 2-*opt*) and the last one (for $k = 7$, 3-*opt*) in our VND implementation. Hernández-Pérez et al. (2009) prove the good performance of 2-*opt* and 3-*opt* as EE local search operators within a VND to solve the 1-PDTSP. As mentioned before and contrary to Hernández-Pérez et al. (2009), we do not deal with unfeasible solutions. In our implementation of these operators, the local search procedures find less-cost solutions while unfeasible paths are discarded.

Both neighborhood structures follow the ideas proposed in Lin (1965) to speed up the search process. For each node i, we store a list of its k nearest neighbors and we sort them in increasing order according to travel costs in c_{ij}. Thus, 2-*opt* operator only scans for possible exchanges between each node i and the k closest nodes to i. Similarly, 3-*opt* procedure, evaluates k interchanges for each one of the k nearest neighbors for each node i. Therefore, the complexity of our 2-*opt* and 3-*opt* are $\mathcal{O}(|\mathcal{N}| \cdot k)$ and $\mathcal{O}(|\mathcal{N}| \cdot k^2)$, respectively. Finally, let us comment that since the number of nodes may vary significantly, adequate values for k depend on $|\mathcal{N}|$. Large values for k would lead to a scenario in which $k = |\mathcal{N}|$ and the complexity becomes $\mathcal{O}(|\mathcal{N}|^2)$ and $\mathcal{O}(|\mathcal{N}|^3)$ for 2-*opt* and 3-*opt*, respectively.

$Or-opt(\lambda)$ After 2-*opt* exploration, our VND calls $Or-opt(\lambda)$ neighborhoods. $Or-opt$ operators are firstly described in Or (1976), and then mentioned in Babin et al. (2007) as one of the best known CE improvement heuristics for the TSP. It aims to improve a solution by first moving a chain of λ consecutive nodes to a different location in the solution path. Moreover, $Or - opt$ heuristic also allows to firstly reverse chains and then move them as described. Our VND includes four variations on $Or - opt(\lambda)$ setting $\lambda = \{2, 3\}$ and then, reversing chains in both cases. From now, we refer to these movements as $Or-opt(2)$, $Or-opt(3)$, $Or-opt_r(2)$, and $Or-opt_r(3)$, respectively. Thereby, these four combinations are neighborhoods two to five of our algorithm ($k = \{2, 3, 4, 5\}$).

Given the structure of our VND and particularly, the use of function `Reverse`, we adapt $Or - opt(\lambda)$ operators only to check for feasible and improvement movements in which chains are moved to previous positions in the path. Therefore, intermediate nodes are shifted forward λ positions. The complexity for each one of the four $Or - opt(\lambda)$ movements is $\mathcal{O}(|\mathcal{N}|^2)$.

Move backward This is the sixth neighborhood within our VND ($k = 6$). This operator attempts to find better solutions by moving a node from its original position to a previous one in the path. Note that this operator can be seen as a special case of our $Or - opt(\lambda)$ movements if $\lambda = 1$ and therefore, the complexity of move backward operator remains as $\mathcal{O}(|\mathcal{N}|^2)$.

Finally, as mentioned before, we set an order to evaluate them within the VND: (i) 2-opt, (ii) $Or - opt(2)$, (iii) $Or - opt(3)$, (iv) $Or - opt_r(2)$, (v) $Or - opt_r(3)$, (vi) move backward and (vii) 3-opt. This particular order established for the neighborhoods relies on their complexity. As Resende and Ribeiro (2016) point out, and appropriate order can save a significant amount of computation time. Thus, small neighborhoods may be explored first while more complex or large neighborhoods are evaluated later. In the case of our VND, the complexity of all the neighborhoods from (i) to (vi) is $\mathcal{O}(|\mathcal{N}|^2)$ while 3-*opt* is computed in $\mathcal{O}(|\mathcal{N}|^3)$.

4.1.3 Perturbation

Algorithm 3 depicts the proposed perturbation function which is mainly based on the 2-*opt* operator described in Sect. 4.1.2. Our function applies a 2-*opt* movement on np sequences of nodes from a solution s. For each sequence, i.e. $\sigma_{ij} = (s_i, s_{i+1}, \ldots, s_{j-1}, s_j)$, the starting point i is chosen in a random fashion between the first position in s and the position $|\mathcal{N}| - \beta$ where β is the size of the sequence to change (line 4). It is worth to mention that the 2-*opt* operator used in the perturbation function (line 5) only searches the first feasible movement. Thus, our perturbation strategy does not check whether the objective function improves and a perturbed solution s_p may end up with a higher value for the objective function (i.e. $f(s_p) > f(s)$). Let us comment that large values for np and β would lead to a significant increase in computational time. Then, small values are desirable and are shown later in Sect. 5.3. Nontheless, if no feasible movement is found along the np sequences, the perturbation procedure delivers the initial solution s.

Algorithm 3 Perturbation

1: **function** PERTURBATION (s, np, β)
2: $s_p \longleftarrow s$
3: **for** $i = 1$ to np **do**
4: Select at random $r \in [1, |\mathcal{N}| - \beta]$
5: $s_p \longleftarrow$ `2-opt`$(s_p, r, r + \beta)$
6: **end for**
7: **return** s_p
8: **end function**

4.2 Multi-start iterated local search

ILS was introduced by Lourenço et al. (2003) as a hybrid metaheuristic based on a heuristic composed by a constructive algorithm, an improvement strategy as local search and a perturbation function. ILS metaheuristic improves a solution s by calling a local search algorithm. Then, an iterated process is started where the incumbent solution is perturbed and improved by a local search algorithm on each iteration. Solution s is replaced by the local optima solution at the end of each iteration in case of improvement. An ILS with more than one initial solution is called MS-ILS. Using the notation introduced in Sect. 4.1, MS-ILS and ILS can be seen as special cases of MS-ELS in Algorithm 1 where `MaxChildren` $= 1$ for MS-ILS, and `MaxStarts` $= 1$ and `MaxChildren` $= 1$ for ILS.

4.3 Greedy randomized adaptive search procedure (GRASP)

Feo and Resende (1995) define GRASP as a multi-start metaheuristic that consists of two steps: construction and improvement (e.g. local search procedure). While the first one aims to build a solution, the second one finds a local optimum using a local search algorithm. After a fixed number of constructive solutions, GRASP ends and returns the best overall solution. Note that GRASP can also be described as a special case of MS-ELS metaheuristic in which `MaxStarts` > 1, `MaxIterations` $= 0$ and `MaxChildren` $= 0$ in Algorithm 1. In order to show the impact of having these values for parameters `MaxIterations` and

`MaxChildren` or those as described in Sect. 4.2, a comparison between GRASP, MS-ILS and MS-ELS is performed in Sect. 5.3.

5 Computational experiments

The computational experiments presented in this section are based on the MILP described in Sect. 3 and the MS-ELS algorithm depicted in Algorithm 1. We firstly describe the sets of instances we use in the computational experiments. Then, we show the results solving the 1-PDTSP MILP (Palacio and Rivera 2019). This model was solved via commercial solver (Gurobi 8.1) setting a maximum computation time of 3600 s for each instance. Lastly, we describe the results obtained via MS-ELS without a fixed maximum computation time. The mathematical model and the metaheuristic algorithm were coded in Visual C++ for Windows 10 running on an Intel Core i7 at 2.70 GHz with 8.00 gigabytes of RAM.

5.1 Data sets

To test the performance of our solution strategy on 1-PDTSP, we solved two different data sets. The first one is a set of benchmark instances previously reported in the literature. On the other hand, and since the BRP is a well-known application of the 1-PDTSP, we also test our algorithm on instances generated using data from the operation of EnCicla, the BSS in the Aburrá Valley (Antioquia, Colombia).

5.1.1 Benchmark instances

The benchmark instances are available at http://hhperez.webs.ull.es/PDsite/. These instances are classified in two sets. The first one, composed of small number of nodes with $|\mathcal{N}| \in \{20, 30, 40, 50, 60\}$ and the second set with large instances in which $|\mathcal{N}| \in \{100, 200, 300, 400, 500\}$. Node demands vary from -10 to 10 (i.e., $q_i \in [-10, 10]\ \forall\ i \in \mathcal{N}$). In this section, we set the vehicle capacity to ten, being the smallest possible value to find feasible solutions and therefore, the hardest configuration to solve (Hernández-Pérez et al. 2009). For each size of the problem, ten instances are available named from A to J. Thus, 100 instances were solved. For a detailed description about the instances generation, we refer the reader to Hernández-Pérez and Salazar-González (2004a) and Hernández-Pérez and Salazar-González (2004b).

5.1.2 EnCicla BSS instances

The Aburrá Valley (Antioquia, Colombia) is a region located in the south central part of Antioquia department in Colombia. This valley is composed by ten urban areas from north to south: Barbosa, Girardota, Copacabana, Bello, Medellín, Envigado, Itagüí, Sabaneta, La Estrella and Caldas. Since 1980, *Área metropolitana del Valle de Aburrá* (AMVA) is the public entity responsible for planning and management on some common policies of Aburrá Valley territory as transportation, environmental policies, among others.

AMVA as authority on mobility and transportation management is the main sponsor of EnCicla, the public bicycle sharing system in Aburrá Valley. EnCicla began its operation in 2011 with six stations and 105 bikes. Later, in 2013, the system increased the number of stations and bicycles to 13 and 420, respectively. In 2017, 52 stations were available for

users and currently, the system is under expansion again and a total of 100 new stations are under construction and 1000 new bikes will be added to the system. Bicycles from EnCicla are available for users from Monday to Friday, starting at 5:30 in the morning to 22:00. On Saturdays, the system operates from 6:30 to 16:00. Nowadays, EnCicla has more than 80,000 active users.

In order to test the performance of our solution strategy using real data from a BSS, we designed a set of instances based on EnCicla. The information required to build the instances was provided by *Subdirección de Movilidad* department (SMD) in AMVA for the operation of EnCicla for 36 days in March and April, 2017 when 52 stations served the system. SMD provided us with the number of bikes available at the beginning and end of the BSS operation in each station for each one of the 36 days. Thus, we were able to compute for 35 days, the number of bikes picked up or delivered at each station during the night repositioning operation. To do so, let us define b_i^t and e_i^t as the number of bikes available at station i before starting and after finishing operations at day t, respectively. Then, the number of bikes (q_i^t) to pickup or deliver at station i at day t is calculated as:

$$q_i^t = e_i^{t-1} - b_i^t, \qquad \forall\, i \in \mathcal{N}\backslash\{0\},\ t = 1, \ldots, T \tag{14}$$

where T denotes the number of days to analyze (i.e., number of instances to solve). Values for q_0^t are calculated as showed before in Eq. (10) to ensure flow conservation throughout the system. We used google maps services to get the geographical position for each one of the 52 stations and then, calculate an euclidean distance between each pair (i, j) of them (c_{ij}). Finally, we fixed the vehicle capacity to 45 (i.e. the largest capacity for a vehicle in EnCicla fleet).

5.2 Results on MILP

Tables 1, 2, 3 and 4 summarize the results obtained when solving the 1-PDTSP benchmark instances using the MILP presented in Sect. 3. Firstly, for the 30 instances with up to 40 nodes, it is possible to find the optimal solution in less than 3600 s. Thus, Table 1 reports for each instance, the value for the objective function (column Opt.) and the time in seconds that the solver required to prove optimality. Secondly, for the 20 instances with 50 and 60 nodes, the solver finds the optimal solution in eight cases (i.e. 40% of the instances). Columns LB and UB report the lower and upper bound reported by the solver, respectively. Column gap, computed as $(UB - LB)/UB$ shows that this ratio is not greater than 5.50% and 11.70% for 50 and 60 nodes instances, respectively. Instances with 100 nodes are presented in Table 3. For each one of the ten instances it was possible to find a feasible solution but no optimality proof was delivered by the optimizer. For this size of instances, gap is always less than 30% and around 20% on average. Lastly, for instances in which $|\mathcal{N}|$ is greater or equal than 200, we report in Table 4 lower bounds since the solver is not able to find any feasible solution in less than 3600 s. Moreover, let us comment that there are five cases in which the optimizer does not report lower bounds in less than 3600. Nonetheless, we set a naive estimation of this bound (NLB) for instances B, D, F, G and I with 400 nodes as:

$$NLB = \sum_{i \in \mathcal{N}} \min_{j:(i,j)\in\mathcal{A}} \{c_{ij}\} \tag{15}$$

For the BSS instances from EnCicla operation, Table 5 reports the lower bound, upper bound, gap and computational time required to solve each instance. Similarly to benchmark instances, the column gap in Table 5 is computed as $(UB - LB)/UB$. Thus, the optimizer

Table 1 Computational results on MILP for 1-PDTSP instances with $|\mathcal{N}| \leq 40$

$\|\mathcal{N}\| = 20$			$\|\mathcal{N}\| = 30$			$\|\mathcal{N}\| = 40$		
Instance	Opt.	Time (s)	Instance	Opt.	Time (s)	Instance	Opt.	Time (s)
n20q10A	4963	7.48	n30q10A	6403	101.64	n40q10A	7173	1034.12
n20q10B	4976	1.45	n30q10B	6603	9.08	n40q10B	6557	661.14
n20q10C	6333	8.23	n30q10C	6486	80.44	n40q10C	7528	222.30
n20q10D	6280	2.19	n30q10D	6652	21.85	n40q10D	8059	1675.57
n20q10E	6415	4.11	n30q10E	6070	3.30	n40q10E	6928	1574.83
n20q10F	4805	1.86	n30q10F	5737	5.20	n40q10F	7506	1635.10
n20q10G	5119	1.01	n30q10G	9371	187.46	n40q10G	7624	842.43
n20q10H	5594	2.10	n30q10H	6431	6.91	n40q10H	6791	733.63
n20q10I	5130	6.02	n30q10I	5821	21.50	n40q10I	7215	140.12
n20q10J	4410	1.29	n30q10J	6187	25.27	n40q10J	6512	212.75
Avg.		3.57			46.26			873.20

Table 2 Computational results on MILP for 1-PDTSP instances with $|\mathcal{N}| \in \{50, 60\}$

$\|\mathcal{N}\| = 50$					$\|\mathcal{N}\| = 60$				
Instance	LB	UB	Time (s)	Gap (%)	Instance	LB	UB	Time (s)	Gap (%)
n50q10A	6987	6987	619.84	0.00	n60q10A	8190	8653	3600.00	5.35
n50q10B	9488	9488	2721.35	0.00	n60q10B	8514	8514	2957.56	0.00
n50q10C	8913	9110	3600.00	2.16	n60q10C	9141	9462	3600.00	3.39
n50q10D	10,085	10,294	3600.00	2.03	n60q10D	10,650	11,243	3600.00	5.27
n50q10E	9492	9492	2190.77	0.00	n60q10E	9224	9487	3600.00	2.77
n50q10F	8398	8887	3600.00	5.50	n60q10F	8388	9499	3600.00	11.70
n50q10G	7126	7126	582.89	0.00	n60q10G	8565	9153	3600.00	6.42
n50q10H	8545	9019	3600.00	5.26	n60q10H	8424	8424	2083.92	0.00
n50q10I	8329	8329	3523.04	0.00	n60q10I	8869	9524	3600.00	6.88
n50q10J	8456	8456	249.89	0.00	n60q10J	8236	9138	3600.00	9.87
Avg.			2428.78	1.49				3384.15	5.16

reports the optimal solution for 28 out of 35 instances obtaining a maximum gap of 5.29% (see instance 6) and also a gap of 0.34% on average. Let us remark that vehicle capacity and traveling costs are constant for the whole set of 35 instances. Hence, only demand on stations differentiate each instance. Results in Table 5 show that objective function (UB) values are very sensitive to changes in stations demand since UB takes values from 2534 to 3440. This variation implies an increase in UB up to 35% with respect to its minimum value. Similarly, changes in units to pick up and deliver at each station have an impact on the computation time required to solve instances. While instance 34 is solved to optimality in 4.22 s, 1 h is not enough to close the gap for the instance 6 (5.29% of gap). This allow us to conclude that CPU times required to solve EnCicla instances via commercial solver are also highly sensitive to variations on demands.

Table 3 Computational results on MILP for 1-PDTSP instances with $|\mathcal{N}| = 100$

Instance	LB	UB	Gap (%)
n100q10A	11,021	14,206	22.42
n100q10B	12,179	15,277	20.28
n100q10C	13,118	17,506	25.07
n100q10D	13,488	17,822	24.32
n100q10E	10,955	13,194	16.97
n100q10F	10,900	12,736	14.42
n100q10G	11,052	15,751	29.83
n100q10H	11,976	13,680	12.46
n100q10I	13,072	15,141	13.66
n100q10J	12,357	15,865	22.11
Avg.			20.15

Table 4 Computational results on MILP for 1-PDTSP instances with $|\mathcal{N}| \geq 200$

$\|\mathcal{N}\| = 200$		$\|\mathcal{N}\| = 300$		$\|\mathcal{N}\| = 400$		$\|\mathcal{N}\| = 500$	
Instance	LB	Instance	LB	Instance	LB	Instance	LB
n200q10A	14,954	n300q10A	17,416	n400q10A	23,112	n500q10A	20,387
n200q10B	15,530	n300q10B	17,420	n400q10B	10,612[a]	n500q10B	19,259
n200q10C	14,010	n300q10C	16,374	n400q10C	20,939	n500q10C	22,456
n200q10D	18,033	n300q10D	19,443	n400q10D	10,517[a]	n500q10D	22,155
n200q10E	16,215	n300q10E	20,776	n400q10E	18,021	n500q10E	22,144
n200q10F	17,999	n300q10F	18,779	n400q10F	10,282[a]	n500q10F	20,735
n200q10G	14,883	n300q10G	18,151	n400q10G	10,348[a]	n500q10G	19,314
n200q10H	18,032	n300q10H	16,606	n400q10H	18,463	n500q10H	26,866
n200q10I	15,245	n300q10I	18,456	n400q10I	9885[a]	n500q10I	22,348
n200q10J	16,273	n300q10J	16,929	n400q10J	18,879	n500q10J	22,335

[a]Naive lower bound (NLB) computed as in Eq. (15)

5.3 Results on MS-ELS

In this section, we present the results based on the MS-ELS strategy. As mentioned before, as MS-ELS is a generalization of GRASP and MS-ILS, we also vary the values of `parametersMaxStarts`, `MaxIterations` and `MaxChildren` to run MS-ILS and GRASP. Moreover, since running times of MS-ELS, MS-ILS and GRASP are roughly proportional to the number of calls to the VND (Rivera et al. 2013), we fixed a number of 300 calls to control the execution time and fairly compare the algorithms performance. After testing several configurations for `MaxStarts`, `MaxIterations`, and `MaxChildren` within MS-ELS and MS-ILS, we found the best results on average with values reported in Table 6. Let us recall that for GRASP, `MaxIterations` and `MaxChildren` are always set to zero. Table 6 also shows the final values for parameters required in the construction phase, VND scheme, 2-*opt* and 3-*opt* operators and perturbation function.

Table 7 presents the computational results for the set of small instances (i.e. $|\mathcal{N}| \leq 60$). In this table, we show the name of the instance and column Opt reports the value of the objective

Table 5 Computational results on MILP for EnCicla instances

Instance	LB	UB	Gap (%)	Time (s)
1	2737	2737	0.00	194.43
2	2769	2769	0.00	115.25
3	2781	2781	0.00	1315.59
4	2722	2722	0.00	400.55
5	3074	3074	0.00	3556.01
6	3258	3440	5.29	3600.00
7	3002	3068	2.15	3600.00
8	2746	2746	0.00	961.45
9	2803	2803	0.00	1731.30
10	2760	2760	0.00	69.45
11	2838	2838	0.00	944.08
12	2723	2723	0.00	47.54
13	2648	2648	0.00	19.56
14	2936	2936	0.00	2223.79
15	2621	2621	0.00	308.82
16	2793	2793	0.00	155.91
17	2777	2818	1.45	3600.00
18	2891	2891	0.00	3254.71
19	2684	2684	0.00	58.03
20	2569	2569	0.00	24.20
21	2791	2817	0.92	3600.00
22	2859	2859	0.00	3443.89
23	2794	2794	0.00	75.66
24	2653	2653	0.00	35.50
25	2878	2878	0.00	865.95
26	2595	2595	0.00	178.14
27	2874	2888	0.48	3600.00
28	2536	2536	0.00	5.09
29	2781	2781	0.00	74.07
30	2574	2574	0.00	10.44
31	2793	2793	0.00	2433.94
32	2852	2852	0.00	819.73
33	2784	2799	0.54	3600.00
34	2534	2534	0.00	4.22
35	2752	2782	1.08	3600.00
Avg.			0.34	1386.50

function for the optimal solution; these values were retrieved from our results in Sect. 5.2 and from Hernández-Pérez et al. (2009). For each instance, we compare five different methods for the 1-PDTSP and BRP: the hybrid GRASP/VND in Hernández-Pérez et al. (2009), the GA in Zhao et al. (2009) and our three strategies, GRASP, MS-ILS and MS-ELS. For each strategy, columns # opt. show the number of times the optimal solution was found over ten runs (for the GRASP/VND this information is not available in Hernández-Pérez et al. (2009)). Columns

Table 6 Parameter values for solution strategy

	MS-ELS	MS-ILS	GRASP
Solution framework			
MaxStarts	5	15	300
MaxIterations	12	20	0
MaxChildren	5	1	0
Greedy randomized construction			
φ	10		
VND			
h_{min}	1		
h_{max}	2		
2-*opt* and 3-*opt*			
k	$2 \cdot \sqrt{\lvert\mathcal{N}\rvert}$		
Perturbation			
np	4		
β	6		

Avg. display the average value for the objective function and columns gap (%) show the difference between the average objective function retrieved (z_{Avg}) and the optimal solution reported in columns Opt (z^*). This gap is computed as follows:

$$gap(\%) = \frac{z_{Avg} - z^*}{z^*} \times 100 \tag{16}$$

For instances in which $|\mathcal{N}| = 20$, the five strategies deliver the optimal solution in each one of the ten runs. Similarly, GA and our three solution algorithms are able to provide the optimal solution for instances with 30 nodes in all runs. For instances with $|\mathcal{N}| = \{40, 50, 60\}$, note that gaps delivered by MS-ELS are smaller than those reported by the other four methods. Moreover, the number of times that MS-ELS finds the optimal solution is greater or equal that the number of optimal values obtained with GRASP and MS-ILS, except for instance n60q10I. Moreover, there exist improvements on average solutions if the evolutionary component is added to GRASP and ILS algorithms. It is worth to mention that quality on solutions found by GA (Zhao et al. 2009) is higher than the one delivered by GRASP/VND from Hernández-Pérez et al. (2009) and our GRASP and MS-ILS. However, MS-ELS outperforms GA and GRASP/VND algorithms. Note also that our GRASP is able to find better solutions on average than the hybrid GRASP/VND from Hernández-Pérez et al. (2009). This result allow us to conclude that $Or - opt(\lambda)$ operators and function `Reverse` within the VND help significantly finding better local optimum solutions for small instances. Let us comment also, that Hernández-Pérez et al. (2009) define $k = 4 \cdot \sqrt{|\mathcal{N}|}$ for 2-*opt* and 3-*opt* neighborhood size while MS-ELS runs with a fixed value for $k = 2 \cdot \sqrt{|\mathcal{N}|}$ and therefore, less computational effort is required for these local search operators.

In Table 8, we summarize the results on large instances. Due to the outperforming behavior of MS-ELS over GRASP and MS-ILS for the smaller instances, we only compare the evolutionary local search strategy with the GRASP/VND (Hernández-Pérez et al. 2009) and GA (Zhao et al. 2009). For each one of the strategies, we report in columns Best, the minimum value for objective function found by the algorithm (bold numbers denote the best solution found among the three algorithms). To the best of our knowledge, optimal solu-

Table 7 Computational results for small instances

Instance	Opt.[c]	GRASP/VND[a]		GA[b]			GRASP			MS-ILS			MS-ELS		
		Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)
$\|\mathcal{N}\| = 20$															
n20q10A	4963	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00
n20q10B	4976	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00
n20q10C	6333	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00
n20q10D	6280	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00
n20q10E	6415	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00
n20q10F	4805	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00
n20q10G	5119	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00
n20q10H	5594	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00
n20q10I	5130	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00
n20q10J	4410	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00
Average			0.00			0.00			0.00			0.00			0.00
$\|\mathcal{N}\| = 30$															
n30q10A	6403	6406.8	0.06	10	6403.0	0.00	10	6403.0	0.00	10	6403.0	0.00	10	6403.0	0.00
n30q10B	6603	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00
n30q10C	6486	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00
n30q10D	6652	6655.1	0.05	10	6652.0	0.00	10	6652.0	0.00	10	6652.0	0.00	10	6652.0	0.00
n30q10E	6070	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00
n30q10F	5737	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00
n30q10G	9371	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00
n30q10H	6431	6431.2	0.00	10	6431.0	0.00	10	6431.0	0.00	10	6431.0	0.00	10	6431.0	0.00
n30q10I	5821	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00
n30q10J	6187	6187.4	0.01	10	6187.0	0.00	10	6187.0	0.00	10	6187.0	0.00	10	6187.0	0.00
Average			0.01			0.00			0.00			0.00			0.00

Table 7 continued

Instance	Opt.[c]	GRASP/VND[a]		GA[b]			GRASP			MS-ILS			MS-ELS		
		Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)
$\|\mathcal{N}\| = 40$															
n40q10A	7173	7188.5	0.22	8	7179.0	0.08	9	7175.4	0.03	8	7182.7	0.14	10	7173.0	0.00
n40q10B	6557	6568.5	0.18	5	6564.5	0.11	4	6566.4	0.14	6	6563.8	0.10	7	6561.5	0.07
n40q10C	7528	7528.4	0.01	10	7528.0	0.00	5	7529.6	0.02	7	7528.6	0.01	10	7528.0	0.00
n40q10D	8059	8135.6	0.95	8	8075.4	0.20	4	8097.8	0.48	5	8095.0	0.45	10	8059.0	0.00
n40q10E	6928	6959.3	0.45	10	6928.0	0.00	3	6941.2	0.19	3	6931.2	0.05	10	6928.0	0.00
n40q10F	7506	7590.5	1.13	10	7506.0	0.00	4	7544.1	0.51	5	7538.3	0.43	10	7506.0	0.00
n40q10G	7624	7682.8	0.77	10	7624.0	0.00	3	7660.6	0.48	6	7645.3	0.28	10	7624.0	0.00
n40q10H	6791	6795.7	0.07	10	6791.0	0.00	5	6801.0	0.15	9	6793.7	0.04	10	6791.0	0.00
n40q10I	7215	7219.0	0.06	8	7215.2	0.00	4	7220.9	0.08	6	7217.5	0.03	10	7215.0	0.00
n40q10J	6512	6513.3	0.02	10	6512.0	0.00	10	6512.0	0.00	10	6512.0	0.00	10	6512.0	0.00
Average			0.38			0.04			0.21			0.15			0.01
$\|\mathcal{N}\| = 50$															
n50q10A	6987	6996.7	0.14	10	6987.0	0.00	1	6993.7	0.10	6	6989.6	0.04	10	6987.0	0.00
n50q10B	9488	9512.6	0.26	8	9501.8	0.15	2	9513.0	0.26	3	9497.9	0.10	10	9488.0	0.00
n50q10C	9110	9133.7	0.26	1	9119.5	0.10	1	9128.2	0.20	2	9143.9	0.37	5	9113.5	0.04
n50q10D	10,260	10,464.3	1.99	2	10,354.8	0.92	1	10,458.1	1.93	3	10,428.5	1.64	5	10,265.4	0.05
n50q10E	9492	9625.1	1.40	7	9574.5	0.87	1	9656.0	1.73	3	9589.3	1.03	10	9492.0	0.00
n50q10F	8684	8773.2	1.03	8	8692.5	0.10	1	8741.4	0.66	4	8735.4	0.59	10	8684.0	0.00
n50q10G	7126	7217.4	1.28	9	7133.5	0.11	1	7230.5	1.47	4	7178.9	0.74	10	7126.0	0.00
n50q10H	8885	9006.5	1.37	1	8956.9	0.81	2	9007.9	1.38	1	9062.1	1.99	3	8893.2	0.09
n50q10I	8329	8412.5	1.00	7	8357.5	0.34	1	8422.9	1.13	3	8391.2	0.75	3	8344.1	0.18
n50q10J	8456	8666.1	2.48	1	8475.8	0.23	0	8595.1	1.64	2	8529.3	0.87	10	8456.0	0.00
Average			1.12			0.36			1.05			0.81			0.04

Table 7 continued

Instance	Opt.[c]	GRASP/VND[a]		GA[b]			GRASP			MS-ILS			MS-ELS		
		Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)	# Opt	Avg.	Gap (%)
$\|\mathcal{N}\| = 60$															
n60q10A	8602	8726.6	1.45	5	8634.8	0.38	1	8719.9	1.37	1	8709.3	1.25	10	8602.0	0.00
n60q10B	8514	8683.2	1.99	10	8514.0	0.00	2	8604.8	1.07	2	8593.9	0.94	10	8514.0	0.00
n60q10C	9453	9565.6	1.19	3	9485.5	0.34	1	9582.8	1.37	1	9585.3	1.40	1	9479.8	0.28
n60q10D	11,059	11,320.6	2.37	1	11,140.2	0.73	1	11,282.5	2.02	1	11,296.0	2.14	1	11,121.6	0.57
n60q10E	9487	9724.8	2.51	1	9592.1	1.11	0	9614.7	1.35	1	9615.0	1.35	5	9494.7	0.08
n60q10F	9063	9437.2	4.13	1	9192.2	1.43	1	9292.2	2.53	1	9221.3	1.75	3	9115.6	0.58
n60q10G	8912	9107.9	2.20	1	8996.0	0.94	1	9082.0	1.91	1	9059.9	1.66	1	8955.8	0.49
n60q10H	8424	8467.3	0.51	3	8472.3	0.57	1	8460.2	0.43	1	8458.4	0.41	10	8424.0	0.00
n60q10I	9394	9529.6	1.44	1	9505.8	1.19	2	9502.2	1.15	1	9519.5	1.34	1	9452.9	0.63
n60q10J	8750	8956.5	2.36	1	8803.3	0.61	1	8894.1	1.65	1	8913.6	1.87	3	8788.7	0.44
Average			2.01			0.73			1.48			1.41			0.31

[a]Results taken from Hernández-Pérez et al. (2009)
[b]Results taken from Zhao et al. (2009)
[c]Results taken from Hernández-Pérez et al. (2009)

tions have not been reported in the literature for these instances. Columns Avg. and SD show the average and standard deviation for objective function. Standard deviation values are not available in Hernández-Pérez et al. (2009) for the GRASP/VND. Finally, we also present the differences on solution quality between our solution strategy and the GRASP/VND and GA in columns gap GRASP/VND and gap GA, respectively. As example, we compute gap GA as follows:

$$gapGA(\%) = \frac{Best_{MS-ELS} - Best_{GA}}{Best_{GA}} \times 100 \tag{17}$$

where $Best_{MS-ELS}$ and $Best_{GA}$ are the best solution found by MS-ELS and GA, respectively. Gap GRASP/VND is computed in a similar way. Lastly, in Table 8, we also show column gap LB in which the lower bound (LB) obtained via MILP and $Best_{MS-ELS}$ value are compared:

$$gapLB(\%) = \frac{Best_{MS-ELS} - LB}{Best_{MS-ELS}} \times 100 \tag{18}$$

Our evolutionary local search strategy is able to find better solutions than GA for 46 out of 50 large instances (i.e. 92% of the instances) and outperforms GRASP/VND in all cases. Note also that standard deviation for the MS-ELS over ten runs is less than the reported in Zhao et al. (2009) for the GA, except for instances in which $|\mathcal{N}| = 300$. Nevertheless, for this ten instances, our MS-ELS delivers better solutions than the genetic algorithm. Moreover, it is worth to comment that as the number of nodes increases, the average differences between MS-ELS and the other algorithms (i.e. gap GRASP/VND and gap GA), also increase. This is, the solution quality of our strategy on average, becomes higher than the one provided by other methods if $|\mathcal{N}|$ is larger. It can be seen also, if the average solution provided by the MS-ELS is compared with the best solution delivered by GRASP/VND and GA. In Table 9, we present the percentage of instances in which average objective function value found via MS-ELS is less than the best solution reported using other two strategies. For the largest subset of instances ($|\mathcal{N}| = 500$) our average solutions outperform the best solution found via GRASP/VND and GA.

Regarding computational times, we also compare MS-ELS performance with GRASP/VND and GA. In Table 10, we present the average CPU time (in seconds) required to solve each subset of large instances. We also show the time needed to construct solutions as well as the time that the MS-ELS requires to improve solutions via VND. In general, MS-ELS requires more computational effort to find solutions with a better solution quality. Nevertheless, let us comment that our strategy is able to find on average a similar solution quality within computational times as those reported in Hernández-Pérez et al. (2009) and Zhao et al. (2009). Figure 5 depicts for instance n100q10A, the evolution of the objective function value for the best solution found. Note that values for objective function solving the 1-PDTSP via MS-ELS are similar for GRASP/VND and GA when computational times coincide with those reported in Table 10 for instances with 100 nodes. In addition, our MS-ELS continues improving solution until the end of time horizon (28 s), which indicates that it can find even better solutions if more time is given.

Table 11 summarizes the results obtained on EnCicla instances solving the 1-PDTSP via MS-ELS. In this table, we show a comparison between our metaheuristic strategy and the exact approach based on MILP in Sect. 3. For each instance, we report in column Best, the minimum value found by the MS-ELS for the objective function as well as the number of times this solution is delivered after 10 runs of the MS-ELS (column # best). Then, we compare the value in column Best and the lower bound reported by the optimizer after the MILP is solved (see Table 5) and we report whether the best solution found is the optimal. Average for the objective function value over the ten runs is also computed (column Avg).

Table 8 Computational results for large instances

	GRASP/VND[a]		GA[b]			MS-ELS					
	Best	Avg.	Best	Avg.	SD	Best	Avg.	SD	Gap GRASP/VND (%)	Gap GA(%)	Gap LB(%)
$\|\mathcal{N}\| = 100$											
n100q10A	11,874	12,087.6	11,828	11,922.6	71.3	**11,760**	11,834.7	41.5	−0.96	−0.57	6.28
n100q10B	13,172	13,582.6	13,114	13,301.6	157.1	**12,938**	13,084.1	65.0	−1.78	−1.34	5.87
n100q10C	14,063	14,421.3	13,977	14,095.2	147.2	**13,958**	13,991.2	24.5	−0.75	−0.14	6.02
n100q10D	14,490	14,787.5	**14,253**	14,406.4	111.9	14,297	14,407.7	69.7	−1.33	0.31	5.66
n100q10E	11,546	12,502.6	**11,411**	11,436.4	52.4	11,419	11,503.2	69.9	−1.10	0.07	4.06
n100q10F	11,734	12,010.7	11,644	11,699	34.5	**11,613**	11,732.5	50.2	−1.03	−0.27	6.14
n100q10G	12,049	12,366.9	12,038	12,120.2	104.8	**11,889**	11,972.3	39.9	−1.33	−1.24	7.04
n100q10H	12,892	13,169.2	12,818	12,906.2	125.1	**12,742**	12,799.1	30.8	−1.16	−0.59	6.01
n100q10I	14,048	14,390.2	14,032	14,137.2	95.9	**13,799**	13,918.6	63.5	−1.77	−1.66	5.27
n100q10J	13,430	13,737.6	13,297	13,516.8	216.4	**13,240**	13,402.9	73.5	−1.41	−0.43	6.67
Average					111.7			52.8	−1.26	−0.59	5.90
$\|\mathcal{N}\| = 200$											
n200q10A	18,013	18,564	17,686	17,987	201.9	**17,642**	17,749.1	57.0	−2.06	−0.25	15.24
n200q10B	18,154	18,932.5	17,798	18,069.4	243.1	**17,393**	17,888.2	232.5	−4.19	−2.28	10.71
n200q10C	16,969	17,280.3	16,466	16,751.2	245.8	**16,430**	16,563.3	113.5	−3.18	−0.22	14.73
n200q10D	21,565	22,285.7	21,306	21,564.4	207.3	**21,244**	21,520.6	153.0	−1.49	−0.29	15.11
n200q10E	19,913	20,643.2	**19,299**	19,713	358.9	19,422	19,582.8	117.8	−2.47	0.64	16.51
n200q10F	21,949	22,284.6	21,910	22,144	247.7	**21,536**	21,649.2	101.9	−1.88	−1.71	16.42
n200q10G	17,956	18,627.7	17,712	17,797.8	80.6	**17,564**	17,721.3	107.4	−2.18	−0.84	15.26
n200q10H	21,463	22,084.9	21,276	21,584	278.4	**19,921**	21,219.3	480.9	−7.18	−6.37	9.48
n200q10I	18,606	19,184.8	18,380	18,509.8	149.6	**18,068**	18,415.2	213.9	−2.89	−1.70	15.62
n200q10J	19,273	19,839.5	18,970	19,274.2	205.5	**18,763**	19,224.1	208.8	−2.65	−1.09	13.27
Average					221.9			178.7	−3.02	−1.41	14.24

Table 8 continued

	GRASP/VND[a]		GA[b]			MS-ELS					
	Best	Avg.	Best	Avg.	SD	Best	Avg.	SD	Gap GRASP/VND (%)	Gap GA(%)	Gap LB(%)
$\|\mathcal{N}\| = 300$											
n300q10A	23,244	24,052.9	23242	23592	265.1	**22973**	23172.6	176.5	−1.17	−1.16	24.19
n300q10B	23,187	23,845.6	22934	23028.6	114.9	**22779**	23011.6	145.9	−1.76	−0.68	23.53
n300q10C	21,800	22,516.6	21922	22083.4	189.6	**21029**	21774.6	429.8	−3.54	−4.07	22.14
n300q10D	25,971	26,462.1	25883	26289.8	253.5	**25448**	25664.3	186.8	−2.01	−1.68	23.60
n300q10E	27,420	27,892.1	27367	27923.8	358.5	**26412**	26994.3	446.4	−3.68	−3.49	21.34
n300q10F	24,852	25,278.2	24826	25055.4	171.8	**23507**	24391.3	589.8	−5.41	−5.31	20.11
n300q10G	24,308	24,760.5	23868	24300.6	412.0	**23632**	23907.8	184.5	−2.78	−0.99	23.19
n300q10H	22,684	23,116.5	21625	21965	278.5	**21513**	22031.6	289.6	−5.16	−0.52	22.81
n300q10I	24,633	25,492.6	24513	24959.2	330.1	**24112**	24697.1	391.0	−2.12	−1.64	23.46
n300q10J	23,086	23,530.2	22810	23045	351.1	**22796**	23097.3	201.4	−1.26	−0.06	25.74
Average					272.5			304.2	−2.89	−1.96	23.01
$\|\mathcal{N}\| = 400$											
n400q10A	31,486	31,912	31,678	31,964.4	309.9	**30,522**	30,977.2	257.7	−3.06	−3.65	24.28
n400q10B	24,883	25,606.4	24,262	24,752.4	283.2	**24,226**	24,711.4	196.5	−2.64	−0.15	56.20
n400q10C	28,942	29,463.2	28,741	29,287.4	603.6	**28,405**	28,580.0	121.6	−1.86	−1.17	26.28
n400q10D	24,597	25,308.6	24,508	24,794.8	320.1	**23,604**	24,223.8	306.3	−4.04	−3.69	55.44
n400q10E	25,548	26,120	25,071	25,473	276.4	**24,497**	25,087.4	330.1	−4.11	−2.29	26.44
n400q10F	27,169	27,755.1	26,681	27,362.8	411.6	**26,409**	26,959.8	315.2	−2.80	−1.02	61.07
n400q10G	24,626	25,088.4	**23,891**	24,290.4	273.0	24,052	24,287.6	159.2	−2.33	0.67	56.98
n400q10H	26,030	26,468.8	25,348	25,811.4	351.5	**25,245**	25,496.9	194.2	−3.02	−0.41	26.86
n400q10I	28,992	29,596.6	28,714	29,261.6	488.7	**28,172**	28,659.3	272.6	−2.83	−1.89	64.91
n400q10J	26,204	26,916.2	26,010	26,489.4	281.6	**25,494**	25,711.3	191.9	−2.71	−1.98	25.95
Average					360.0			234.5	−2.94	−1.56	42.44

Table 8 continued

	GRASP/VND[a]		GA[b]			MS-ELS					
	Best	Avg.	Best	Avg.	SD	Best	Avg.	SD	Gap GRASP/VND (%)	Gap GA(%)	Gap LB(%)
$\|\mathcal{N}\| = 500$											
n500q10A	28,742	29,323.6	28,857	29,258.8	478.3	**27,923**	28,218.9	265.1	−2.85	−3.24	26.99
n500q10B	27,335	27,711.1	26,648	27,454.8	525.7	**26,309**	26,596.6	181.5	−3.75	−1.27	26.80
n500q10C	31,108	31,692.7	30,701	31,426.8	609.4	**29,787**	30,469.1	336.6	−4.25	−2.98	24.61
n500q10D	30,794	31,428.4	30,994	31,442.2	376.9	**30,017**	30,200.1	231.6	−2.52	−3.15	26.19
n500q10E	30,674	31,371.7	30,905	31,154.6	231.3	**28,731**	29,897.1	497.7	−6.33	−7.03	22.93
n500q10F	28,957	29,812.3	28,882	29,241	244.9	**28,112**	28,540.6	353.1	−2.92	−2.67	26.24
n500q10G	27,198	27,958.2	27,107	27,473	212.5	**26,519**	26,738.8	197.0	−2.50	−2.17	27.17
n500q10H	36,857	37,361.1	37,626	38,142.4	258.8	**35,855**	36,154.2	223.5	−2.72	−4.71	25.07
n500q10I	31,045	31,536	30,796	31,044.6	306.0	**29,713**	30,239.3	293.7	−4.29	−3.52	24.79
n500q10J	31,412	31,877.9	31,255	32,310	617.9	**30,028**	30,618.1	328.2	−4.41	−3.93	25.62
Average					386.2			290.8	−3.65	−3.47	25.64

[a]Results taken from Hernández-Pérez et al. (2009)
[b]Results taken from Zhao et al. (2009)

Table 9 Average solution for MS-ELS versus best solutions for GRASP and GA

	100 (%)	200 (%)	300 (%)	400 (%)	500 (%)
GRASP/VND	100	100	80	100	100
GA	40	20	50	50	100

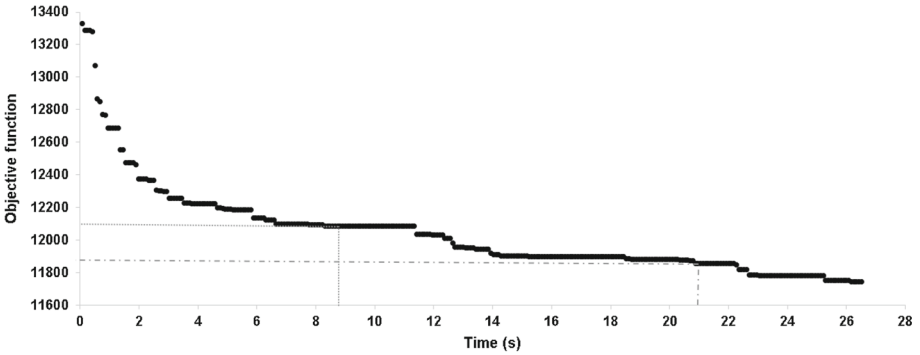

Fig. 5 Objective function value versus time (s) for instance n100q10A

Table 10 CPU times (s) comparison for large instances

$\|\mathcal{N}\|$	GRASP/VND[a]	GA[b]	MS-ELS		
			Construction	VND	Total
100	8.85	21.12	0.81	27.21	28.02
200	41.76	95.23	3.54	110.75	114.29
300	117.86	212.59	9.82	279.14	288.96
400	220.4	358.22	24.97	650.01	674.98
500	391.47	570.15	39.67	927.96	967.63

[a]Results taken from Hernández-Pérez et al. (2009)
[b]Results taken from Zhao et al. (2009)

Since in Table 5, a lower and upper bound (LB and UB, respectively) are available, we show gaps between the best solution delivered by the MS-ELS and these bounds. Thus, columns Min. gap in Table 11 are computed as follows: Min. $\text{gap}_{LB} = Best - LB/Best$ and Min. $\text{gap}_{UB} = Best - UB/Best$, regarding lower and upper bound values from MILP, respectively. Average gaps over the ten runs are also reported in columns Avg. gap. Finally, the average CPU time required to solve each instance via MS-ELS is also showed.

Note that MS-ELS is able to find the optimal solution for 28 out of the 35 EnCicla instances. For the remaining seven instances, in which MS-ELS do not deliver the optimal value for the objective function, the metaheuristic finds at least the same upper bound reported by the optimizer when MILP is solved. For instances 6, 7 and 35, the MS-ELS improves the upper bound found via commercial solver when the MILP is solved. In these cases, column Min. gap_{UB} shows negative values. Lastly, let us remark that our MS-ELS is able to find minimum gap of 0.27% on average, when solutions are compared with LB values.

Table 11 Computational results on MS-ELS for EnCicla instances

Instance	Best	# Best	Opt.	Avg.	MILP LB		MILP UB		Avg. time (s)
					Min. gap_{LB} (%)	Avg. gap (%)	Min. gap_{UB} (%)	Avg. gap (%)	
1	2737	3	*	2775.57	0.00	1.38	0.00	1.38	12.85
2	2769	4	*	2790.60	0.00	0.77	0.00	0.77	14.69
3	2781	2	*	2807.20	0.00	0.93	0.00	0.93	14.30
4	2722	7	*	2730.57	0.00	0.31	0.00	0.31	15.20
5	3074	8	*	3080.67	0.00	0.21	0.00	0.21	17.12
6	3364	1		3415.93	3.15	4.61	− 2.26	− 0.72	13.32
7	3066	3		3100.23	2.09	3.16	− 0.07	1.03	13.03
8	2746	4	*	2762.83	0.00	0.61	0.00	0.61	16.12
9	2803	5	*	2865.13	0.00	2.15	0.00	2.15	14.01
10	2760	6	*	2766.57	0.00	0.23	0.00	0.23	15.86
11	2838	6	*	2853.90	0.00	0.55	0.00	0.55	15.17
12	2723	4	*	2734.20	0.00	0.41	0.00	0.41	14.41
13	2648	7	*	2650.83	0.00	0.11	0.00	0.11	15.74
14	2936	3	*	2942.10	0.00	0.21	0.00	0.21	14.66
15	2621	5	*	2628.77	0.00	0.29	0.00	0.29	16.17
16	2793	6	*	2806.40	0.00	0.47	0.00	0.47	17.07

Table 11 continued

Instance	Best	# Best	Opt.	Avg.	MILP LB		MILP UB		Avg. time (s)
					Min. gap_{LB} (%)	Avg. gap (%)	Min. gap_{UB} (%)	Avg. gap (%)	
17	2818	3		2850.43	1.45	2.55	0.00	1.11	13.59
18	2891	7	*	2894.33	0.00	0.11	0.00	0.11	14.87
19	2684	6	*	2692.53	0.00	0.31	0.00	0.31	12.91
20	2569	7	*	2575.97	0.00	0.27	0.00	0.27	16.15
21	2817	3		2832.00	0.92	1.45	0.00	0.53	14.97
22	2859	2	*	2868.30	0.00	0.32	0.00	0.32	15.11
23	2794	4	*	2844.90	0.00	1.76	0.00	1.76	14.37
24	2653	6	*	2660.57	0.00	0.28	0.00	0.28	17.05
25	2878	6	*	2887.60	0.00	0.33	0.00	0.33	14.73
26	2595	5	*	2608.53	0.00	0.51	0.00	0.51	16.29
27	2888	4		2912.40	0.48	1.31	0.00	0.83	13.35
28	2536	1	*	2781.13	0.00	2.56	0.00	2.56	19.29
29	2781	3	*	2824.10	0.00	1.52	0.00	1.52	16.34
30	2574	6	*	2597.80	0.00	0.90	0.00	0.90	17.29
31	2793	2	*	2838.47	0.00	1.60	0.00	1.60	14.72
32	2852	1	*	2892.53	0.00	1.40	0.00	1.40	14.40
33	2799	3		2810.33	0.54	0.94	0.00	0.40	14.76
34	2534	7	*	2544.40	0.00	0.40	0.00	0.40	14.83
35	2777	2		2792.67	0.90	1.45	− 0.18	0.38	15.94
Avg.		4.34		2812.01	0.27	1.04	− 0.07	0.70	15.16

Table 12 Computational results for EnCicla instances with warm start on MILP

Instance	MILP (with warm start)				MILP		Δ gap(%)	Δ time (%)
	UB	LB	Gap (%)	Time (s)	Gap (%)	Time (s)		
1	2737	2737	0.00	307.19	0.00	194.43	0.00	− 58.00
2	2769	2769	0.00	142.01	0.00	115.25	0.00	− 23.22
3	2781	2781	0.00	1859.57	0.00	1315.59	0.00	− 41.35
4	2722	2722	0.00	417.00	0.00	400.55	0.00	− 4.11
5	3074	3074	0.00	2717.02	0.00	3556.00	0.00	23.59
6	3258	3361	3.06	3600.00	5.29	3600.00	2.23	0.00
7	3006	3066	1.96	3600.00	2.15	3600.00	0.19	0.00
8	2746	2746	0.00	650.15	0.00	961.45	0.00	32.38
9	2803	2803	0.00	641.21	0.00	1731.30	0.00	62.96
10	2760	2760	0.00	53.98	0.00	69.45	0.00	22.28
11	2838	2838	0.00	920.17	0.00	944.08	0.00	2.53
12	2723	2723	0.00	28.53	0.00	47.54	0.00	39.99
13	2648	2648	0.00	14.43	0.00	19.56	0.00	26.23
14	2936	2936	0.00	1498.38	0.00	2223.79	0.00	32.62
15	2621	2621	0.00	92.69	0.00	308.82	0.00	69.99
16	2793	2793	0.00	128.85	0.00	155.91	0.00	17.36
17	2796	2818	0.78	3600.00	1.45	3600.00	0.67	0.00
18	2891	2891	0.00	1094.39	0.00	3254.71	0.00	66.38
19	2684	2684	0.00	26.65	0.00	58.03	0.00	54.08
20	2569	2569	0.00	3.82	0.00	24.20	0.00	84.21
21	2784	2817	1.17	3600.00	0.92	3600.00	− 0.25	0.00
22	2831	2860	1.01	3600.00	0.00	3443.89	− 1.01	− 4.53
23	2794	2794	0.00	43.71	0.00	75.66	0.00	42.23
24	2653	2653	0.00	19.32	0.00	35.50	0.00	45.58
25	2878	2878	0.00	1010.72	0.00	865.95	0.00	− 16.72
26	2595	2595	0.00	36.66	0.00	178.14	0.00	79.42
27	2888	2888	0.00	2470.97	0.48	3600.00	0.48	31.36
28	2536	2536	0.00	4.04	0.00	5.09	0.00	20.63
29	2781	2781	0.00	18.71	0.00	74.07	0.00	74.74
30	2574	2574	0.00	7.52	0.00	10.44	0.00	27.97
31	2793	2793	0.00	1917.14	0.00	2433.94	0.00	21.23
32	2852	2852	0.00	2246.05	0.00	819.73	0.00	− 174.00
33	2788	2799	0.39	3600.00	0.54	3600.00	0.14	0.00
34	2534	2534	0.00	3.30	0.00	4.22	0.00	21.80
35	2777	2777	0.00	3029.92	1.08	3600.00	1.08	15.84
Avg.			0.24	1228.69	0.34	1386.49	0.10	16.96

The reader may notice that there exist limitations if a real-scenario repositioning operation is treated as a 1-PDTSP. These limitations are derived from two main facts to remark. Firstly, in the static version of the BRP, stations are served during the night or while bike demands are negligible. Secondly, in the 1-PDTSP, a single vehicle is available to visit all the locations. Then, only small size BSS can be served since services times (load and unload bikes) and traveling times between stations are also included in the operation for one vehicle. Our main motivation to study EnCicla instances is not related to test the 1-PDTSP in large bike repositioning operation instances. We aim to evaluate the performance and solution quality of our strategy on a different set of instances in which demands, and node locations were not generated randomly. In real scenarios, for BSS with a large number of stations, the repositioning operation using a single vehicle would lead to non-practical problems.

When solving MILP for EnCicla instances (see Table 5) it is possible to find optimal solutions for 28 out of 35 instances in less than 1 h and an average gap of 0.34%. These results evidence an adequate performance for the exact strategy. However, we conduct an additional experiment for this subset of instances in order to reduce the CPU time required and improve solution quality for instances with no optimality proof. To do so, we use the solution delivered by MS-ELS as a warm start for our MILP. This warm start (the final solution of MS-ELS), is read by the solver as an upper bound once the *branch-and-bound* process begins. Table 12 summarizes the main results obtained with this solution strategy. For each instance, we report the upper and lower bound from the optimizer (see columns UB and LB, respectively). Thus, the gap is computed as $(UB - LB)/UB \times 100$. The CPU time required in seconds to solve the mathematical model (MS-ELS times are already reported in Table 11) is also shown. In Table 12, we recall gap and CPU time if a warm start is not available. We do so, in order to easily display for the reader variations on performance if warm start is used. Column Δ time shows the variation on CPU time if a start solution is available. This variation is computed as Δ time $= (\text{time}_{MILP} - \text{time}_{WS})/\text{time}_{MILP} \times 100$ where time_{MILP} and time_{WS} are the CPU times required to solve the mathematical model avoiding and including the initial solution, respectively. Moreover, in column Δ gap, we show the improvement on gap between solving the MILP without a start solution and the warm start strategy: $\Delta\text{gap} = \text{gap}_{MILP} - \text{gap}_{WS}$ where gap_{MILP} and gap_{WS} are the gaps reported by the optimizer when solving MILP including and skipping the initial solution, respectively.

Results in Table 12 allow to conclude that using MS-ELS as warm start for the MILP improves the mathematical model performance, on average. CPU time improves 16.96% while gap decreases 0.10% over the 35 instances and 0.44% over the eight instances with a variation on gap. Computational time may improve up to 84.21% if an upper bound is computed for MILP (instance 20). Warm start also helps to find new optimality proofs (instances 27 and 35) and reduce gaps even if no optimal solution is reported within the maximum computational times (e.g. instances 6, 7 and 17). Finally, the reader may notice that there are cases in which warm start deteriorates computational times for MILP (e.g. instances 1, 2, 3 and 32) and also gap (instances 21 and 22).

6 Concluding remarks

In this paper, we described a MS-ELS for the 1-PDTSP. In the evolutionary local search strategy, construction of solutions is performed via a greedy randomized algorithm, and then are improved within a VND algorithm. This improvement phase is based on seven

well-known local search operators for the traveling salesman problem. We also described a GRASP and an ILS algorithm as particular cases of MS-ELS. The obtained results allow us to conclude that MS-ELS outperforms two of the algorithms reported in the literature to deal with the 1-PDTSP. Moreover, for small instances, it is possible to evidence the benefits if the evolutionary component is added to GRASP and ILS algorithms. MS-ELS is able to find better solutions than those reported via GRASP and MS-ILS for instances with 40 and 50 nodes. MS-ELS finds optimal solutions on instances with up to 60 nodes and it also delivers better solutions for large instances up to 500 nodes in which optimal solutions are not reported so far.

From a practical perspective, the 1-PDTSP is a simplifying representation of the static BRP. Nevertheless, rebalacing decisions can be made for bike-sharing operations as 1-PDTSPs in small and medimum size BSSs if a single vehicle is able to serve station requeriments. For large systems, mathematical models and solution strategies for the 1-PDTSP as our MS-ELS, can be easily integrated to a clustering strategy in order to find several routes (one per cluster) if multiple vehicles are required. Additionally, considering relevant information from BSS (e.g. service times at stations) is straighforward in 1-PDTSP formulations and solution strategies. However, future research directions include an extension of this work based on multi-vehicle contexts. Some interesting approaches are based on *cluster first, route second* strategies in which several clusters must be defined before routing decisions are made. Additionally, synchronization constraints can be included in order to deal, for example with a heterogeneous fleet problem in which large vehicles are not allowed to visit some locations due to mobility and parking constraints. Matheuristic approaches are also a extension for this work and for the multi-vehicle version of the problem.

Acknowledgements The present research work has been supported by Universidad EAFIT. The authors would like to thank *Subdirección de Movilidad* department from *Área metropolitana del Valle de Aburrá*, for providing us with information for the instances described in Sect. 5.1.2.

References

Babin, G., Deneault, S., & Laporte, G. (2007). Improvements to the or-opt heuristic for the symmetric travelling salesman problem. *Journal of the Operational Research Society*, *58*(3), 402–407.

Belenguer, J. M., Benavent, E., Labadi, N., Prins, C., & Reghioui, M. (2010). Split-delivery capacitated arc-routing problem: Lower bound and metaheuristic. *Transportation Science*, *44*(2), 206–220.

Caggiani, L., & Ottomanelli, M. (2013). A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems. *Procedia-Social and Behavioral Sciences*, *87*, 203–210.

Chemla, D., Meunier, F., & Calvo, R. W. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, *10*(2), 120–146.

Contardo, C., Morency, C., & Rousseau, L. M. (2012). *Balancing a dynamic public bike-sharing system* (Vol. 4). Canada: Cirrelt Montreal.

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

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

Duhamel, C., Lacomme, P., Quilliot, A., & Toussaint, H. (2011). A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem. *Computers and Operations Research*, *38*(3), 617–640.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*(2), 109–133.

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

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, *130*(3), 449–467.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423–454.

Hernández-Pérez, H., & Salazar-González, J. J. (2004a). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, *145*(1), 126–139.

Hernández-Pérez, H., & Salazar-González, J. J. (2004b). Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, *38*(2), 245–255.

Hernández-Pérez, H., Rodríguez-Martín, I., & Salazar-González, J. J. (2009). A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers and Operations Research*, *36*(5), 1639–1645.

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

Ho, S. C., & Szeto, W. (2017). A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. *Transportation Research Part B: Methodological*, *95*, 340–363.

Kloimüllner, C., Papazek, P., Hu, B. & Raidl, G. R. (2014). Balancing bicycle sharing systems: An approach for the dynamic case. In *European conference on evolutionary computation in combinatorial optimization* (pp. 73–84). Berlin: Springer.

Legros, B. (2019). Dynamic repositioning strategy in a bike-sharing system; how to prioritize and how to rebalance a bike station. *European Journal of Operational Research*, *272*(2), 740–753.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, *44*(10), 2245–2269.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In F. Glover & G. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 320–353). Berlin: Springer.

Miller, C. E., Tucker, A. W., & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, *7*(4), 326–329.

Mladenović, N., Urošević, D., Ilić, A., et al. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, *220*(1), 270–285.

Or, I. (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Ph.D. Thesis, Department of Industrial Engineering and Management Science, Northwestern University.

Palacio, J. D. & Rivera, J. C. (2019). Mixed-integer linear programming models for one-commodity pickup and delivery traveling salesman problems. In *Workshop on engineering applications* (pp. 735–751). Springer.

Prins, C. (2009). A GRASP × evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem* (pp. 35–53). Springer.

Raviv, T., Tzur, M., & Forma, I. A. (2013). Static repositioning in a bike-sharing system: Models and solution approaches. *EURO Journal on Transportation and Logistics*, *2*(3), 187–229.

Resende, M. G., & Ribeiro, C. C. (2016). *Optimization by GRASP: Greedy randomized adaptive search procedures* (1st ed.). New York: Springer.

Rivera, J. C., Afsar, H. M. & Prins, C. (2013). Multistart evolutionary local search for a disaster relief problem. In *International conference on artificial evolution (evolution artificielle)* (pp. 129–141). Springer.

Shui, C., & Szeto, W. (2018). Dynamic green bike repositioning problem: a hybrid rolling horizon artificial bee colony algorithm approach. *Transportation Research Part D: Transport and Environment*, *60*, 119–136.

Szeto, W., Liu, Y., & Ho, S. C. (2016). Chemical reaction optimization for solving a static bike repositioning problem. *Transportation Research Part D: Transport and Environment*, *47*, 104–135.

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L., & Velasco, N. (2010). GRASP/VND and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots. *Engineering Applications of Artificial Intelligence*, *23*(5), 780–794.

Wolf, S. & Merz, P. (2007). Evolutionary local search for the super-peer selection problem and the p-hub median problem. In *International workshop on hybrid metaheuristics* (pp. 1–15). Springer.

Zhang, D., Yu, C., Desai, J., Lau, H., & Srivathsan, S. (2017). A time-space network flow approach to dynamic repositioning in bicycle sharing systems. *Transportation Research Part B: Methodological*, *103*, 188–207.

Zhao, F., Li, S., Sun, J., & Mei, D. (2009). Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers and Industrial Engineering*, *56*(4), 1642–1648.