# The Bike sharing Rebalancing Problem with Stochastic Demands

Mauro Dell'Amico[a], Manuel Iori[a], Stefano Novellani[a,*], Anand Subramanian[b]

[a]*DISMI, Università di Modena e Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy*
[b]*Departamento de Sistemas de Computação, Centro de Informática, Universidade Federal da Paraíba, Rua dos Escoteiros, Mangabeira, João Pessoa – PB, 58058-600, Brazil*

**Abstract**

In this paper we deal with the stochastic version of a problem arising in the context of self-service bike sharing systems, which aims at determining minimum cost routes for a fleet of homogeneous vehicles in order to redistribute bikes among stations. The *Bike sharing Rebalancing Problem with Stochastic Demands* is a variant of the one-commodity many-to-many pickup and delivery vehicle routing problem where demands at each station are represented by random variables, with associated probability distributions, that depend on stochastic scenarios. We develop stochastic programming models that are solved using different approaches, in particular, the L-Shaped and branch-and-cut. Moreover, we also propose heuristic algorithms based on an innovative use of positive and negative correlations among stations' stochastic demands, as well as an efficient strategy for checking feasibility. The proposed solution approaches are evaluated and compared by means of extensive computational experiments on newly realistic benchmark instances.

*Keywords:* Stochastic Programming, Vehicle Routing, Bike Sharing, Branch-and-cut, Correlations.

*Declaration of interest: none*

## 1. Introduction

Bike sharing systems are public or private systems used to provide an additional mean of transportation to users moving across city centers. They appeared for the first time in Amsterdam in the 1960s and they have multiplied consistently in the last two decades in every

*Corresponding author

*Email addresses:* mauro.dellamico@unimore.it (Mauro Dell'Amico), manuel.iori@unimore.it (Manuel Iori), stefano.novellani@unimore.it (Stefano Novellani), anand@ci.ufpb.br (Anand Subramanian)

continent. As a matter of fact, according to the *Bike sharing World Map* (see Meddin and DeMaio (2018)), in February 2018 there were more than 1500 active bike sharing systems across the globe.

Bike sharing systems consist of a set of bikes and several stations, located in different places of the city. Each station has a number of slots where users can collect or return a bike. Generally, users start and finish their trips in two different stations, and need not necessarily to perform the return trip using the same mean of transportation. For example, a typical situation arises when the starting station is located on the top of a hill and the ending one is at the bottom. Furthermore, the target number of bikes of a station defines the *balanced level of occupation*. Note that if all the stations are balanced then the system is balanced. A station may get imbalanced if a user collects a bike from a balanced station and returns it to a different one, thus potentially leading to undesirable situations where a station turns out to be empty or full. Therefore, to increase the efficiency of the system as well as users' satisfaction, each station must be brought from the current situation to its balanced level of occupation. This defines a demand or request of a station. In this paper, we consider stochastic demands representing different scenarios, each one with its corresponding probability distribution.

The main goal of this paper is to propose efficient solution methods for the *Bike sharing Rebalancing Problem with Stochastic Demands*, which consists of determining minimum cost routes for a fleet of homogeneous vehicles in order to redistribute bikes among stations with stochastic demands so as to keep the system balanced.

The main contributions of the paper are:

- A new problem: the stochastic version of the deterministic Bike sharing Rebalancing Problem (considered by Dell'Amico et al. (2014)).
- The design and evaluation of five exact approaches, namely: deterministic equivalent program, one-cut and multi-cut L-shaped method, and one-cut and multi-cut single stage branch-and-cut methods. We show that some of these methods are able to solve large instances to optimality.
- A heuristic algorithm based on correlation among stochastic variables.

The remainder of the paper is structured as follows. Section 2 presents a literature review

on the related problems. Section 3 formally defines the problem, whereas stochastic programming models are provided in Sections 4. Section 5 suggests some interesting properties and valid inequalities for checking feasibility that are used to speed up the performance of the algorithms. Section 6 explains the exact algorithms based on the L-shaped and branch-and-cut methods. Section 7 describes the heuristic algorithms based on correlations among stations. Computational results are reported in Section 8. Finally, Section 9 concludes.

## 2. Literature review

Bike sharing systems have been receiving considerable attention from the literature during the past decade. Several associated optimization problems were suggested aimed at rebalancing bike sharing systems. Rebalancing operations can be performed during the night, i.e., when the system is closed or the usage is negligible, or during the day, i.e., when the status of the system changes rapidly. The former case is known as the static rebalancing problem, whereas the latter is known as the dynamic rebalancing problem.

Among the first authors, Chemla et al. (2013) and Raviv et al. (2013) defined and solved static rebalancing problems. Chemla et al. (2013) solved a deterministic static single vehicle rebalancing problem where vertices can be visited more than once and split delivery is allowed. The authors proposed branch-and-cut algorithms for two relaxations of the problem and a tabu search algorithm. Raviv et al. (2013) considered a multi-vehicle static bike sharing rebalancing variant, where the objective is to minimize both operational costs and user's dissatisfaction expressed as a penalty. Stations can be visited multiple times, transshippment is allowed, and a maximum duration is imposed to all routes. They proposed two formulations strengthened with valid inequalities and dominance rules, whereas Forma et al. (2015) developed a heuristic procedure to solve the same problem. Ho and Szeto (2017) solved a multi-vehicle repositioning problem based on the same model by proposing a hybrid large neighborhood search. Moreover, Li et al. (2016) considered multiple types of bikes to be redistributed with a single vehicle proposing a mixed-integer linear programming (MILP) model and a hybrid genetic algorithm. In that problem, when a request of a bike type cannot be met, then another type can be used instead, incurring a penalty. Szeto and Shui (2018) studied a single vehicle static bike repositioning problem that determines the routes and the loading and unloading quantities

that minimize the positive deviation from the tolerance of total demand dissatisfaction and the service times. They use a set of loading and unloading strategies embedded in a bee colony algorithm. Other static versions of the problem with multiple vehicles have been proposed by Rainer-Harbach et al. (2015) and Bulhões et al. (2018), while other static problems with one vehicle have been considered by Ho and Szeto (2014), Erdoğan et al. (2014), Erdoğan et al. (2015), Szeto et al. (2016), Li et al. (2016), Cruz et al. (2017)).

In particular, Erdoğan et al. (2014) introduced a variant of the 1-PDTSP called the *static bicycle relocation problem with demand intervals*, where the number of bikes at each station can lie within an interval. Erdoğan et al. (2015) considered a static problem where the same station may be visited multiple times, thus allowing the demand of the station to be split. Other related works that allow splits to be performed, in addition to Chemla et al. (2013), are: Cruz et al. (2017), Salazar-González and Santos-Hernández (2015), Bruck et al. (2017), Benchimol et al. (2011), and Di Gaspero et al. (2013) among others.

Static problems include the *Bike sharing Rebalancing Problem* (BRP) by Dell'Amico et al. (2014). BRP belongs to the wider class of *pickup and delivery problems*, and according to the scheme defined by Berbeglia et al. (2007) it can be classified as a *many-to-many pickup and delivery problem*, where commodities do not have fixed origins and destinations. One of the most relevant examples of this class of problem is the *one-commodity pickup and delivery traveling salesman problem* (1-PDTSP) introduced by Hernández-Pérez and Salazar-González (2004), where a single vehicle is used to meet the pickup and delivery demands associated with a single commodity. In that paper the authors propose a branch-and-cut algorithm to solve the symmetric and asymmetric versions of the problem. An enhanced branch-and-cut with new valid inequalities was later presented in Hernández-Pérez and Salazar-González (2007). In addition, (meta)heuristics were put forward by Hernández-Pérez and Salazar-González (2004), Hernández-Pérez et al. (2009), Wang et al. (2006), Mladenović et al. (2012). We refer the interested reader to the survey by Battarra et al. (2014) on pickup and delivery problems for freight transportation. Dell'Amico et al. (2014) proposed, in their seminal work, four mathematical formulations for the BRP with multiple vehicles that were solved with branch-and-cut algorithms. Later on, Dell'Amico et al. (2016) implemented destroy and repair algorithms for the BRP and for the version with route duration constraints, as well as a branch-and-cut pro-

cedure for the latter. The present work on the BRP with stochastic demands is based on these deterministic problems.

A dynamic version of the BRP where the fleet of vehicles is heterogeneous was considered by Contardo et al. (2012). They put forward time-indexed formulations that are solved via Dantzing-Wolfe and Benders' decomposition based heuristics. Zhang et al. (2017) studied a dynamic version of the repositioning problem, considering inventory level and user arrival forecasting in a time-space network flow model. They propose a MILP formulation and a math-heuristic approach. Shui and Szeto (2017) introduced a dynamic green bike repositioning problem that minimizes the total unmet demand and the fuel and $CO_2$ emission cost of the repositioning vehicle over an operational period for which they proposed a hybrid rolling horizon artificial bee colony algorithm.

To our knowledge, while the deterministic versions of several BRPs have been studied in many recent works, their stochastic counterparts have not been considered yet. Wang and Wang (2013) provided an analysis on bike repositioning strategies by conducting a simulation study where real time and historical data are considered. Raviv et al. (2013) considered stochastic information implicitly in their penalty function. Regue and Recker (2014) solved a dynamic problem, where routing and inventory problems are both accounted and the unknown demand part is estimated using historical data feeding a forecasting model. Saharidis et al. (2014) suggested a mathematical formulation to design the bike sharing infrastructure, deciding the location and the size of the bike stations, incorporating a hourly demand estimation. Schuijbroek et al. (2017) derived bounds on inventory quantities by modeling inventory as a non-stationary Markov chain, and used them in mixed integer programming and constraint programming models.

Stochastic optimization (see, e.g., Birge and Louveaux (2011) and King and Wallace (2012)) has been applied to solve many vehicle routing problems under uncertainty. For example, vehicle routing problems with stochastic demands have been modeled and solved with chance constrained programming (see, e.g., Stewart and Golden (1983)), robust optimization (see, e.g, Bertsimas and Simchi-Levi (1996)), and stochastic programming with recourse (see, e.g., Laporte et al. (2002)). The reader is referred to Gendreau et al. (2014) and Oyola et al. (2016) for detailed surveys on a variety of stochastic vehicle routing problems. Nevertheless, despite

the broad literature on the topic, problems with positive and negative stochastic demands of the same commodity (i.e., one-commodity pickup and delivery problems with stochastic demands) have been only investigated by Louveaux and Salazar-González (2009) and Louveaux and Salazar-González (2014), where they study the one vehicle version and consider the capacity of the vehicle as an output.

## 3. Problem description

The *Bike sharing Rebalancing Problem with Stochastic Demands* (BRPSD) is modeled on a complete digraph $G = (V, A)$, where $V = \{0, 1, \ldots, n\}$ is the set of vertices including the depot, vertex 0, and the subset of $n$ customers $V_0 = V \setminus \{0\}$, and $A$ is the set of arcs between each pair of distinct vertices, each having an associated non-negative cost $c_{ij}, (i, j) \in A$. Let $c_{\min} = \min_{(i,j)\in A}\{c_{ij}\}$ denote the minimum cost of an arc in the digraph. A request $q_j^\omega$, which denotes the difference between the current level of occupation and the balanced one, is given for each vertex $j \in V_0$ and scenario $\omega \in \Omega$, where $\Omega$ is the set of all the possible scenarios and $p^\omega$ is the probability of the scenario $\omega$, with $\sum_{\omega\in\Omega} p^\omega = 1$. Requests can be positive (pickup vertex) or negative (delivery vertex). The quantities collected at pickup vertices can be used to serve the demand of delivery vertices or can be disposed to the depot. If necessary or convenient, some quantities can come from the depot in order to meet the delivery requests and some others can return to the depot at the end of the rebalancing. Stations balanced *a priori* (i.e., those for which $q_j^\omega = 0$) must also be visited.

In the BRPSD, it is not always possible or even convenient to completely meet the requests of certain vertices in some scenarios. Therefore, we model the problem using a novel recourse function allowing the demands to be partially fulfilled at the price of a penalty. This novel feature is aligned with the suggestion provided by Gendreau et al. (2016) in their study on future research directions in stochastic vehicle routing ("What about allowing the demand of customers to be partially fulfilled (at the price of some penalty) if the vehicle's capacity is exceeded by the total demand on its planned route? Such an option might be particularly welcomed in the case where demands are correlated").

When the request is violated the objective function is penalized by incurring a penalty $\varepsilon \cdot c_{\min}$ $(0 \leq \varepsilon \leq 1)$ for each unit of slack and surplus in vertex $j \in V_0$. This penalty is chosen to be

a fraction of the minimum cost of an arc, so as to have a relation with the traveling costs. A relationship between the penalty and the traveling cost is needed to express a uniform objective function. Because of the nature of the problem, we believe that an unmet request should not be very expensive. Indeed, not delivering or collecting a bike does not imply in large hidden costs, opposed to not meeting a customer request in the traditional VRP. Computational insights on how the penalty affects the problem difficulty and model behaviour are provided in Section 8.5.

Slack and surplus quantities are limited by some considerations on the capacity and the balanced level of occupation of each station, $H_j$ and $d_j$, respectively. In other words, one cannot leave more bikes than the capacity of the station and one cannot pickup more bikes than the available number. Moreover, we set a coefficient $\delta$, $0 \leq \delta \leq 1$, in order to have a leverage on the request violation. If $\delta = 0$, we forbid any violation of the requests, which means that the solution must be feasible for all scenarios. On the other hand, if $\delta = 1$ the only limit to the request violation are the number of bikes present in the station and the station capacity. More specifically, we set the maximum slack and surplus quantities for $j \in V_0$ as $\min\{\lceil \delta H_j \rceil, d_j\}$ and $\min\{\lceil \delta H_j \rceil, H_j - d_j\}$, respectively. Furthermore, we also assume that $|q_j^\omega| \leq H_j$ and $|q_j^\omega| \leq Q, j \in V_0, \omega \in \Omega$, where $Q$ is the vehicle capacity. In Sections 8.4 and 8.5 an extensive study regarding values of $\delta$ and $\varepsilon$ is performed.

The BRPSD aims at routing a fleet of identical vehicles of capacity $Q$ available at the depot in order to partially or completely meet the scenario requests, while minimizing the sum of the travel and penalty costs. Each route starts and ends in the depot and each station must be visited exactly once.

## 4. Formulations for the BRPSD

In this section we present a set of formulations to solve the BRPSD. We adopted the terminology used in Birge and Louveaux (2011).

### *4.1. Deterministic equivalent program*

We first introduce the *Deterministic Equivalent Program* (DEP) formulation, which consists of a large model containing all the scenarios, including the decisions to be taken before and after the realization of the stochastic variables. We define variables $x_{ij}, (i,j) \in A$, that can

take value 1 if arc $(i, j)$ is used and 0 otherwise. Variable $f_{ij}^{\omega}$ represents the quantity flowing on the arc $(i, j)$ in scenario $\omega \in \Omega$, whereas $s_j^{\omega-}$ and $s_j^{\omega+}$ are, respectively, the slack and surplus variables used to pay a cost $k$ for each unit of slack and surplus when the flows do not respect the request at vertex $j \in V_0$ in scenario $\omega \in \Omega$.

$$\text{(DEP)} \quad \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + \varepsilon \cdot c_{\min} \sum_{\omega \in \Omega} p^{\omega} \sum_{j \in V_0} (s_j^{\omega-} + s_j^{\omega+}) \tag{1}$$

$$\sum_{i \in V} x_{ij} = 1 \qquad j \in V_0 \tag{2}$$

$$\sum_{i \in V} x_{ji} = 1 \qquad j \in V_0 \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad S \subseteq V_0, S \neq \emptyset, |S| > 2 \tag{4}$$

$$x_{ij} + x_{ji} \leq 1 \qquad i, j \in V_0, i \neq j \tag{5}$$

$$f_{ij}^{\omega} \leq Q x_{ij} \qquad (i, j) \in A, \omega \in \Omega \tag{6}$$

$$\sum_{i \in V} f_{ji}^{\omega} - \sum_{i \in V} f_{ij}^{\omega} = q_j^{\omega} + s_j^{\omega-} - s_j^{\omega+} \qquad j \in V_0, \omega \in \Omega \tag{7}$$

$$s_j^{\omega-} \leq \min\{\lceil \delta H_j \rceil, d_j\} \qquad j \in V_0, \omega \in \Omega \tag{8}$$

$$s_j^{\omega+} \leq \min\{\lceil \delta H_j \rceil, H_j - d_j\} \qquad j \in V_0, \omega \in \Omega \tag{9}$$

$$s_j^{\omega+}, s_j^{\omega-} \geq 0 \qquad j \in V_0, \omega \in \Omega \tag{10}$$

$$f_{ij}^{\omega} \geq 0 \qquad (i, j) \in A, \omega \in \Omega \tag{11}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A. \tag{12}$$

The first component of the objective function (1) aims at minimizing the traveling costs, while the second component incurs a penalty $k$ for every unit of demand that is not fulfilled, multiplied by the probability of the related scenario. Constraints (2) and (3) impose that exactly one arc must enter and leave every customer vertex, respectively. A maximum number of available vehicles can be easily imposed in the model. Constraints (11) and (12) define the domain of variables $x_{ij}$ and $f_{ij}^{\omega}$. Constraints (4)–(5) avoid subtours, whereas constraints (6) state that the flow variables must not exceed the vehicle capacity. Constraints (7) specify that the difference between the flows leaving and entering a vertex must be equal to its request, in a given scenario, corrected by the surplus and slack variables, which in turn are bounded by constraints (8)–(10). In particular, the vehicle cannot collect more bikes than what is available

at the station and the capacity of the station cannot be exceeded if those values are not greater than $\lceil \delta H_j \rceil$, with $0 \leq \delta \leq 1$, $\delta \in \mathbb{R}$. Figure 1 shows an example of a possible balanced level of occupation $d_j$ with respect to a negative or non-negative request, given that $l_j^\omega$ is the current level of occupation of a station $j \in V_0$ in scenario $\omega \in \Omega$. Note that the current level of occupation $l_j^\omega$ is a stochastic variable, but because the balanced level of occupation is fixed for each station, the demand can also be considered a stochastic variable:

$$q_j^\omega = l_j^\omega - d_j, \qquad j \in V_0, \omega \in \Omega. \tag{13}$$

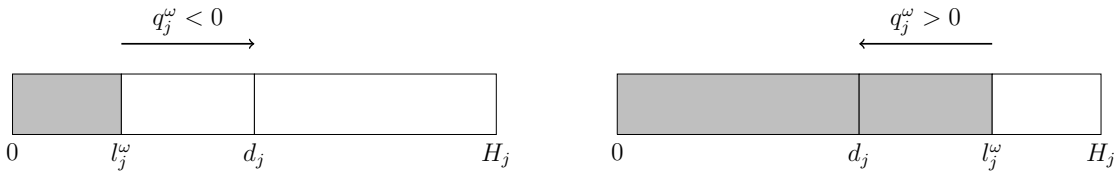


Figure 1: Parameters of a station in case of negative (left) and positive (right) request. In grey the current level of occupation, in white the current empty docks.

### *4.2. Two-stage formulation*

In this section, we propose a *two-stage Stochastic Integer Programming* (SIP) model with simple recourse for the BRPSD. In this case we perform a Benders' decomposition of the DEP formulation into two stages, the first one takes into account the deterministic part of the problem, while the second one represents a set of models (a.k.a. submodels), one for each scenario, that take as input the decisions made at the first stage and make them fit with the request of the various scenarios after the realization of the stochastic variables. This is achieved by generating a cut from the solution of the second stage and adding it to the first stage model as further explained. Both models must be solved in an iterative way: when the inequalities produced by the second stage models do not violate the solution of the first stage then the optimal solution has been found (see, e.g., Birge and Louveaux (2011)).

$$\text{(1-Stage)} \qquad \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + \sum_{\omega \in \Omega} p^\omega z(\omega, s^-, s^+) \tag{14}$$

$$(2) - (5), (12).$$

The objective function (14) of the first stage model contemplates the travel costs and the expected value of the solution of the second stage that defines the costs due to the penalties.

$$\text{(2-Stage)} \qquad \min z(\omega, s^-, s^+) = \varepsilon \cdot c_{\min} \sum_{j \in V_0} (s_j^{\omega-} + s_j^{\omega+}) \tag{15}$$

$$f_{ij}^{\omega} \leq Q\overline{x}_{ij} \qquad (i,j) \in A \tag{16}$$

$$(7) - (11).$$

As for the second stage, the objective function (15) minimizes the penalty costs for the particular scenario $\omega \in \Omega$. Note that $\overline{x}_{ij}, (i,j) \in A$, are fixed values that were obtained by solving the first stage model.

The most common way to solve a two-stage stochastic model is through the L-Shaped method (see, Birge and Louveaux (2011)), which solves the first stage problem (an *Integer Linear Program* (ILP), in our case), takes the linking variables between the two stages ($\overline{x}_{ij}$) and uses them as an input in all the second stage models. The dual problem of the second stage problem is then solved for each $\omega \in \Omega$. We can associate constraints (16) with the dual variables $\xi_{ij}^{\omega}, (i,j) \in A, \omega \in \Omega$, and constraints (7), (8), and (9) with, respectively, the dual variables $\phi_j^{\omega}, \pi_j^{\omega}$, and $\nu_j^{\omega}$, $j \in V_0, \omega \in \Omega$. If at least one of the dual problems of the second stage is unbounded, then the first stage solution is not feasible for this scenario. Therefore, one must add a so-called feasibility cut, which is computed by using the extreme ray of the unbounded dual problem, to the first stage model and solve the latter again. The extreme ray, here indicated as the dual variables with a hat ($\hat{\xi}_{ij}$, $\hat{\phi}_j, \hat{\pi}_j$, and $\hat{\nu}_j$), is a vector of the same size of the solution and it basically expresses the direction of the unboundedness. Hence, the feasibility cut for an infeasible scenario can be written as

$$\begin{aligned} Q \sum_{(i,j) \in A} \hat{\xi}_{ij}^t x_{ij} \geq & - \sum_{j \in V \setminus \{0\}} \hat{\phi}_j^t q_j^t - \sum_{j \in V \setminus \{0\}} \hat{\pi}_j^t \min\{\lceil \delta H_j \rceil, d_j\} \\ & - \sum_{j \in V_0} \hat{\nu}_j^t \min\{\lceil \delta H_j \rceil, H_j - d_j\} \end{aligned} \qquad t \in T_{feas} \tag{17}$$

where $T_{feas}$ is the set that defines the number of feasibility cuts required at the end of the procedure, but keeping in mind that we insert them one at a time. When all submodels are feasible (they are all finite because we have a simple recourse), one may need to introduce a so-called optimality cut, which takes into account the average cost of the second stage problems,

the variable $\eta$ (that considers the expected value of the second stage model costs), and the current solution. In particular, it means that if all second stage problems are feasible, then variable $\eta$ must cover the expected value of the costs of each second stage problem, or that the current solution must be changed if a cheapest option is available. The optimality cut (18) imposes the variable $\eta$ to be greater than or equal to the expected value of the dual solution of the second stage problems, computed for all scenarios. The dual solution is represented by overlined variables ($\overline{\xi}_{ij}^{\omega}$, $\overline{\phi}_{j}^{\omega}$, $\overline{\pi}_{j}^{\omega}$, and $\overline{\nu}_{j}^{\omega}$), whereas $T_{opt}$ is the set representing the amount of optimality cuts required.

$$\eta \geq \sum_{\omega \in \Omega} p^{\omega} \Big( - Q \sum_{(i,j) \in A} \overline{\xi}_{ij}^{\omega,t} x_{ij} - \sum_{j \in V_0} \overline{\phi}_{j}^{\omega,t} q_{j}^{\omega} - \sum_{j \in V_0} \overline{\pi}_{j}^{\omega,t} \min\{\lceil \delta H_j \rceil, d_j\} - \sum_{j \in V_0} \overline{\nu}_{j}^{\omega,t} \min\{\lceil \delta H_j \rceil, H_j - d_j\} \Big) \quad \omega \in \Omega, t \in T_{opt} \tag{18}$$

If the inequality does not violate the current solution then it is optimal, otherwise the optimality cut imposes the solution vector to change or the penalty value $\eta$ to increase. In this latter case we must recompute the first stage model with the new cut.

The resulting formulation of the L-Shaped method, denoted as *Two-stage Formulation* or Benders' Master Problem, can be written as in (19)–(20), where $\eta$ is the variable that takes into account the penalties incurred by the second stage models, as well as the derived feasibility cuts (17) and optimality cuts (18).

$$\text{(Two-stage)} \min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} + \eta \tag{19}$$

$$(2) - (5), (12), (17), (18)$$

$$\eta \geq 0. \tag{20}$$

### *4.3. Multi-cut formulation*

The Multi-cut L-Shaped method (see Birge and Louveaux (2011)) is another way of solving the two-stage problem by inserting, at each iteration for which all scenarios are feasible, one optimality cut for each scenario that needs its component of costs to increase. In this case, we need a variable $\eta^{\omega}$ for each scenario $\omega \in \Omega$, and the summation of these new variables is evaluated in the objective function (21) while the new optimality constraints are represented in

(22). If one scenario is found infeasible a feasibility cut (17) is inserted. The resulting *Multi-cut Formulation* can be written as follows.

$$\text{(Multi-cut)} \min \sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} + \sum_{\omega\in\Omega}\eta^{\omega} \tag{21}$$

$$(2)-(5),(12),(17)$$

$$\eta^{\omega} \geq p^{\omega}\Big( - Q \sum_{(i,j)\in A} \overline{\xi}_{ij}^{\omega,t} x_{ij} - \sum_{j\in V_0} \overline{\phi}_{j}^{\omega,t} q_j^{\omega} - \sum_{j\in V_0} \overline{\pi}_{j}^{\omega,t} \min\{\lceil \delta H_j \rceil, d_j\} - \sum_{j\in V_0} \overline{\nu}_{j}^{\omega,t} \min\{\lceil \delta H_j \rceil, H_j - d_j\}\Big) \quad t \in T_{opt}, \omega \in \Omega \tag{22}$$

$$\eta^{\omega} \geq 0 \qquad \omega \in \Omega. \tag{23}$$

*4.4. Single stage branch-and-cut*

Another possibility to solve the problem is again related to the two-stage method, but a single run of a branch-and-cut algorithm is used. The branch-and-cut is initialized with the 1-Stage model, without the subtour elimination constraints (4). The *CallBack* function of the MIP solver is activated every time the LP solver provides a solution, either fractional or integer. In this implementation the CallBack checks the current solution to find possible violated subtour elimination constraints, violated feasibility cuts, or violated optimality cuts and adds them to the current LP model. The solution found by the branch-and-cut algorithm is the optimal solution of the Benders' Master Problem. Similarly to the previous case, we separated the cuts using the *One-cut* or the *Multi-cut* formulation.

## 5. Properties and valid inequalities for the BRPSD

In this section, we define a set of properties related to feasible paths for the BRPSD built upon the ideas provided in Dell'Amico et al. (2016). Such properties enable one to derive valid inequalities, also described in this section, as well as to speed up the computation of the heuristic procedures described in Section 7.

*5.1. Properties*

The proposed properties are based on the following operators:

- *Remove* a vertex from its current position in the solution;

- *Insert* a vertex in a given position of a route;
- *Move* a vertex from its position to another one (composition of remove and insert);
- *Swap* the position of two vertices (composition of two remove and two insert);
- *Merge* two routes or partial routes by concatenating them, one after the other.

We define a *route* $P$ as an ordered sequence of vertices $\{0\}, P(1), \ldots, P(|P|), \{0\}$, that starts and ends at the depot, where $P(i) \in V_0, i = 1, \ldots, |P|$.

We recall that for each vertex $j \in V_0$ the values $\min\{\lceil \delta H_j \rceil, H_j - d_j\}$ and $\min\{\lceil \delta H_j \rceil, d_j\}$ represent the maximum value that surplus variables $s_j^+$ and slack variables $s_j^-$ can assume, respectively. Reminding that $q_j < 0$ when we leave bikes in $j$ the quantity $\max\{-Q, q_j - \min\{\lceil \delta H_j \rceil, H_j - d_j\}\}$ gives the opposite of the maximum number of bikes we can leave in vertex $j$, taking into account the vehicle and station capacity and the maximum request violation. Similarly $\min\{Q, q_j + \min\{\lceil \delta H_j \rceil, d_j\}\}$ gives the maximum number of bikes we can remove from vertex $j$.

Given a route $P$, for each vertex $P(i) \in P$ in position $i$, and for each scenario $\omega \in \Omega$ let $\lambda^\omega_{P(i)}$ denote the cumulative sum of the opposite of the maximum possible number of bikes, we can leave in each vertex of $P$, up to vertex $P(i)$. Similarly, let $\mu^\omega_{P(i)}$ denote the cumulative sum of the maximum possible number of bikes we can remove/avoid to deliver in each vertex of $P$, up to vertex $P(i)$:

$$\lambda^\omega_{P(i)} = \sum_{j=0}^{i} \max\left\{-Q, q^\omega_{P(j)} - \min\{\lceil \delta H_{P(j)} \rceil, H_{P(j)} - d_{P(j)}\}\right\}, \tag{24}$$

$$\mu^\omega_{P(i)} = \sum_{j=0}^{i} \min\left\{Q, q^\omega_{P(j)} + \min\{\lceil \delta H_{P(j)} \rceil, d_{P(j)}\}\right\}. \tag{25}$$

Moreover, let $\lambda^\omega_0 = \mu^\omega_0 = 0$. By using the values $\lambda$ and $\mu$, we can derive some conditions on the vehicle loads that guarantee the feasibility of route $P$.

Consider the value $\min_{j=0}^{i}\{\lambda^\omega_{P(j)}\}$ and observe that it is non-positive. If this value is strictly negative it means that the route $P$, up to $P(i)$, may be feasible only if the vehicle starts from the depot with an initial load of $|\min_{j=0}^{i}\{\lambda^\omega_{P(j)}\}|$ bikes. Therefore the vehicle load at $P(i)$ must be at least $\lambda^\omega_{P(i)} - \min_{j=0}^{i}\{\lambda^\omega_{P(j)}\}$. Similarly $Q - \max_{j=0}^{i}\{\mu^\omega_{P(j)}\}$ is the greater empty space in the

vehicle, up to $P(i)$, so the maximum possible bike's removal at $P(i)$ is $\mu^{\omega}_{P(i)}+Q-\max^{i}_{j=0}\{\mu^{\omega}_{P(j)}\}$. This leads us to define

$$F^{\omega}_{P(i)} = \left[\lambda^{\omega}_{P(i)} - \min_{j=0}^{i}\{\lambda^{\omega}_{P(j)}\},\ \ \mu^{\omega}_{P(i)} + Q - \max_{j=0}^{i}\{\mu^{\omega}_{P(j)}\}\right], \tag{26}$$

as the *end load window* of $P(i)$, i.e., the feasible interval for the load on the vehicle after leaving vertex $P(i)$ on route $P$, for scenario $\omega \in \Omega$.

**Property 1.** *If the end load windows up to the last customer vertex $P(|P|)$ of a route $P$ are non-empty for all scenarios, then the route is feasible.*

**Proof.** The end load windows $F^{\omega}_{P(i)}$ gives the minimum and maximum load of a vehicle running on route $P$, when leaving vertex $P(i)$. If this load interval is non-empty vertex $P(i)$ can be feasibly served. If the interval is non-empty for all vertices of the route, the entire route is feasible. □

In Figure 2 we provide an example that illustrates Property 1. We have a route $P = (0, 1, 2, 3, 4, 0)$, a fixed scenario and requests $q = (0, 5, -6, -7, 10, 0)$. We also set vehicle capacity $Q = 20$ and, for sake of simplicity, $\min\{\lceil \delta H_j \rceil, H_j - d_j\} = \min\{\lceil \delta H_j \rceil, d_j\} = 2$ for all vertices. The plot reports, in solid line, the cumulative sum of all requests along the route. The dashed line represents $\mu$, while the dotted line represents $\lambda$. One can see that $\min^{i}_{j=0}\{\lambda_{P(j)}\} = (0, 0, -5, -14, -14)$, and $Q - \max^{i}_{j=0}\{\mu_{P(j)}\} = (20, 13, 13, 13, 10)$. Therefore, $F = ([0, 0], [3, 20], [0, 16], [0, 11], [8, 20])$ and all the vertices in the root are feasible, as the entire root.

Similarly, we can define for a vertex $j \in V_0$ the *start load window* $B^{\omega}_{j}$, that represents the load interval which guarantees the feasibility of the route from $P(i)$ to the depot. In this case, we need values similar to $\lambda$ and $\mu$, but in a reverse logic. We denote such values as $\gamma$ and $\zeta$. *** SIamo sicuri che sia $\gamma$ il corrispondente di $\lambda$ ?? Mi pare che le formule siano opposte *** These quantities are the cumulative sums of the loads adjusted by the maximum slack and surplus quantities computed from the last node of the path to the current one, in a backward way. We can now compute the sum of reverse quantities corrected by the maximum surplus and slack

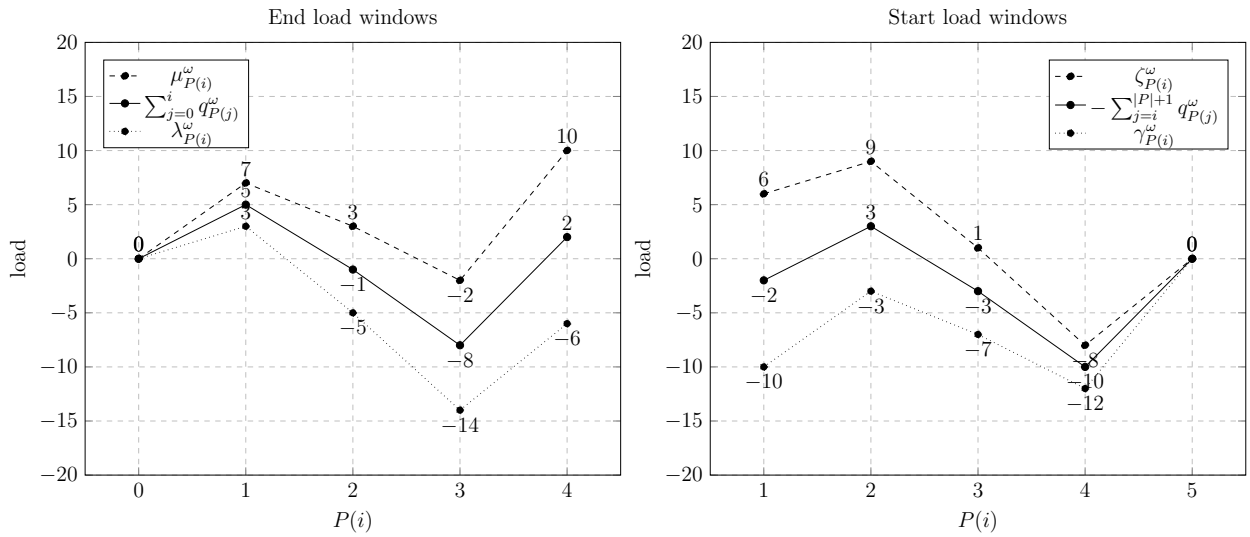


Figure 2: Example of end load windows and start load windows computation.

values on a route $P$, for the vertex $P(i) \in P$ in position $i$ for scenario $\omega \in \Omega$, as follows:

$$\gamma^{\omega}_{P(i)} = -\sum_{j=i}^{|P|+1} \min\left\{Q, q^{\omega}_{P(j)} + \min\{\lceil \delta H_{P(j)} \rceil, d_{P(j)}\}\right\} \tag{27}$$

$$\zeta^{\omega}_{P(i)} = -\sum_{j=i}^{|P|+1} \max\left\{-Q, q^{\omega}_{P(j)} - \min\{\lceil \delta H_{P(j)} \rceil, H_{P(j)} - d_{P(j)}\}\right\}, \tag{28}$$

where we set $\gamma^{\omega}_{P(|P|+1)} = \zeta^{\omega}_{P(|P|+1)} = 0$, $\omega \in \Omega$. We recall that $P(|P|+1)$ is the depot.

Let us define the *start load window*, denoted as $B$, as the interval of loads in which the vehicle load must lie when entering vertex $P(i)$ to ensure the feasibility of route $\{0\}, P(1), \ldots, P(|P|), \{0\}$, where $P(i) \in V_0, i = \ldots, |P|$, for scenario $\omega \in \Omega$, i.e.:

$$B^{\omega}_{P(i)} = \left[\gamma^{\omega}_{P(i)} - \min_{j=i}^{|P|+1}\{\gamma^{\omega}_{P(j)}\},\ \ \zeta^{\omega}_{P(i)} + (Q - \max_{j=i}^{|P|+1}\{\zeta^{\omega}_{P(j)}\})\right]. \tag{29}$$

Similarly to Property 1, we can write the following:

**Property 2.** *If the start load windows from the end, up to vertex $P(1)$ of a route $P$ is non-empty for all scenarios, then the route is feasible.*

In Figure 2, one can see an example of the computation of $\gamma$ and $\zeta$ along a route. For this example we can compute the start load window of the route as described in (29), namely: $B =$

$([xx, xx], [xx, xx], [xx, xx], [xx, xx], [0, 0])$ and thus the route is feasible for the corresponding scenario.

From Properties 1 and 2 we can derive two more properties.

**Property 3.** *Two disjoint and feasible routes $P$ and $R$ can be merged in a feasible route $P \oplus R$, where the last customer vertex of $P$ is followed by the first customer vertex of $R$ if the following condition is satisfied for each scenario $\omega \in \Omega$:*

$$F^{\omega}_{P(|P|)} \cap B^{\omega}_{R(1)} \neq \emptyset.$$

**Property 4.** *Given a feasible route $P$:*

- *Removing the first and last customer vertex from route $P$ is always feasible.*
- *Removing a customer vertex $P(i)$ from route $P$, where $1 < i < |P|$, is feasible if $F^{\omega}_{P(i-1)} \cap B^{\omega}_{P(i+1)} \neq \emptyset$ for all scenarios $\omega \in \Omega$.*
- *Let $R$ denote a route consisting of the single customer $i$. The insertion of vertex $i$ in route $P$ after the vertex $P(p)$ is feasible if the two following conditions are satisfied for every scenario $\omega \in \Omega$: (1) $i$ can be feasibly inserted after the first part of the route $P$, i.e., $F^{\omega}_{P(p)} \cap B^{\omega}_{R(1)} \neq \emptyset$; and (2) the novel first part of the route (path $\{0\}, P(1), P(2), ..., P(p), \{i\}$) can be feasibly merged with the second part of the route $P$, i.e., $F^{\omega}_{i}(P) \cap B^{\omega}_{P(p+1)} \neq \emptyset$.*

### *5.2. Valid inequalities*

Let us consider a route $P = (0, i, j, 0)$ composed of two vertices. Because of Property 1, if $F^{\omega}_{j}$ is an empty interval for at least one scenario $\omega \in \Omega$, then such route is infeasible. Thus, if Property 1 is not satisfied for a given pair of vertices $i, j \in V_0$, then $x_{ij}$ can be removed from the model or one can simply set $x_{ij} = 0$. The preprocessing procedure that performs such verification for each pair of vertices $i, j \in V_0$ is denoted as *RemoveArcs*.

By using a similar rationale, we now introduce two valid inequalities. Given a pair of vertices $i, j \in V_0$, $i \neq j$, let $N^{+}(i, j)$ (resp., $N^{-}(i, j)$) contain those vertices $h \in V_0, h \neq i, j$, associated

with route $(0, i, j, h, 0)$ (resp., $(0, h, i, j, 0)$), that are infeasible for at least one scenario because of Property 1 (resp., Property 2). The following inequalities hold.

$$x_{ij} + \sum_{h \in N^+(i,j)} x_{jh} \leq 1 \qquad i, j \in V_0, \tag{30}$$

$$\sum_{h \in N^-(i,j)} x_{hi} + x_{ij} \leq 1 \qquad i, j \in V_0. \tag{31}$$

In (30) we state that, if a route $(0, i, j, h, 0)$ is infeasible by following Property 1, we must impose that at most one arc is selected between $(i, j)$ and $(j, h)$ in the optimal solution. Moreover, we can strengthen the inequality by including all the arcs $(j, h)$ that imply infeasible routes because of constraints (3), which set to one the maximum cardinality of arcs exiting a node. A similar rationale is used in (31), where if a route $(0, h, i, j, 0)$ is infeasible for Property 2, one must select at most one arc between $(i, j)$ and $(j, h)$ in the optimal solution. We thus strengthen the inequality by including all the arcs $(h, i)$ that imply infeasible routes due to constraints (2). Both inequalities (30) and (31) can be exactly separated in $O(n^2)$ by complete enumeration.

## 6. Exact algorithms

In this section we describe the implementations of the exact approaches proposed for the BRPSD, namely: *DEP*, *One-cut L-Shaped*, *Multi-cut L-Shaped*, *One-cut single stage*, and *Multi-cut single stage*.

### *6.1. First stage, DEP, and single stage implementations*

The ILP associated with the first stage "1-Stage" is solved using a branch-and-cut scheme. The preprocessing procedure (*RemoveArcs*) described in Section 5.2 is applied to eliminate unnecessary variables. Subtour elimination constraints (4) are separated in polynomial time using the well-known procedure by Crowder and Padberg (1980), by selecting the most violated cut. Inequalities (30) and (31) are separated by complete enumeration adding the most violated one at a time, and are applied only in the L-Shaped method implementations where they allow to improve the average gap between the best upper and lower bound by 1%, on the tested instances.

An initial primal bound, obtained using the heuristics presented in Section 7, is provided to the algorithm and the strong branching strategy is adopted.

The *DEP* implementation solves the entire formulation (1)–(12) using the same branch-and-cut scheme employed for the first stage implementation.

The *One-cut* and *Multi-cut single stage* branch-and-cut apply a similar scheme to formulation "1-Stage": *RemoveArcs* preprossesing, the same primal bound, strong branching strategy, and the same separation procedure for the subtour elimination constraints.

### *6.2. L-Shaped method implementations*

We define as *One-cut L-Shaped* the algorithm that solves the Two-stage Formulation "Two-stage" described in Section 4.2, with the L-Shaped method. This method solves the first stage model "1-Stage" to optimality. The first stage, in turn, provides an integer solution that is plugged into the second stage models "2-Stage", one for each scenario. If for at least one scenario the corresponding second stage model is infeasible, the algorithm aborts the resolution of the remaining ones and inserts the derived feasibility cut (17) to the first stage model that is solved again and so on. In case all second stage models are feasible the algorithm computes and inserts the derived optimality cut (18) to the first stage model and reiterates. When no feasibility cut is produced and the optimality cut does not violate the current solution, the optimal solution is obtained.

The *Multi-cut L-Shaped* algorithm that solves the formulation "Multi-cut" of Section 4.3 is similar to the L-Shaped described above; however, when optimality cuts are needed, one optimality cut (22) derived from each one of the second stage models, and thus for each scenario, is inserted in the first stage model that is solved again and the procedure reiterates. When no feasibility cut is produced and the optimality cuts do not violate the current solution we obtain the optimal solution.

## 7. Heuristic algorithms based on correlations

One-commodity pickup and delivery problems, such as the one considered in this paper, make use of the positive requests of some vertices to meet the negative requests of other vertices. If requests are stochastic, solution methods can take advantage of the existing negative and positive correlations between vertices' requests. For example, while the requests of a set of vertices are usually negative for a relevant number of scenarios, they can be typically positive

for another set. Therefore, the correlation between the request of a vertex of the first set and the request of a vertex of the second one will be negative. It is thus reasonable to look for the arcs connecting vertices with negatively correlated requests because coupling them may help to keep the load of the vehicle far from its extremes, i.e., 0 and $Q$, and hence more vertices can be served with the same vehicle. Of course, travel costs contribute substantially to the objective function, and it is also reasonable to look for the cheapest arcs. To our knowledge, the proposed use of correlations among requests is novel in vehicle routing. In the following, for sake of simplicity, we write "correlation between two vertices", or two routes, when referring to the correlation among their stochastic demands.

### *7.1. Constructive procedures*

The two algorithms proposed in this section are based on the well-known nearest neighbor and savings constructive heuristics (see, e.g., Toth and Vigo (2014)), and they both consider travel costs as well as correlations in the evaluation function.

#### *7.1.1. Nearest neighbor with correlations*

We first propose a constructive procedure, called *Nearest neighbor with correlations,* that takes into account not only the travel cost, but also the correlations while evaluating the feasible insertion of an unrouted vertex at the end of a possible partial route, where the correlation is expressed as the Pearson-Bravais coefficient (see, e.g., Pearson (1920)). Given two vertices $i, j \in V_0, i \neq j$, their correlation is computed as the ratio between the covariance $\sigma_{ij}$ and the product of the standard deviations of the two vertices, $\sigma_i$ and $\sigma_j$, namely:

$$\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} = \frac{\sum_{\omega \in \Omega} p^\omega (q_i^\omega - \overline{q}_i)(q_j^\omega - \overline{q}_j)}{\sqrt{\sum_{\omega \in \Omega} p^\omega (q_i^\omega - \overline{q}_i)^2} \sqrt{\sum_{\omega \in \Omega} p^\omega (q_j^\omega - \overline{q}_j)^2}},$$

where $\overline{q}_i = \sum_{\omega \in \Omega} p^\omega q_i^\omega, i \in V_0$ and $-1 \leq \rho_{ij} \leq 1$. If $\rho_{ij} = -1$, then there is a perfect negative correlation between $i$ and $j$, while they are perfectly positively correlated if $\rho_{ij} = 1$. If two vertices $i$ and $j$ are negatively correlated, then there is a good chance that in many scenarios an increase in the request of one vertex can correspond to a decrease in the request of the other vertex, and if not too expensive, the arc $(i, j)$ is likely to be in a high quality solution. Otherwise, positive correlations suggest that two vertices have similar behaviors among the scenarios, and hence they are not likely to be adjacent in a high quality solution.

When building an initial solution with our nearest neighbor heuristic, we consider the correlation between the current route $R$ and the vertex $j$, which is computed as follows:

$$\rho_{Rj} = \frac{\sigma_{Rj}}{\sigma_R \sigma_j} = \frac{\sum_{\omega\in\Omega} p^\omega \left(\sum_{i\in R} q_i^\omega - \overline{q}_R\right)(q_j^\omega - \overline{q}_j)}{\sqrt{\sum_{\omega\in\Omega} p^\omega \left(\sum_{i\in R} q_i^\omega - \overline{q}_R\right)^2}\sqrt{\sum_{\omega\in\Omega} p^\omega (q_j^\omega - \overline{q}_j)^2}}, \tag{32}$$

where $\overline{q}_R = \sum_{\omega\in\Omega} p^\omega \sum_{i\in R} q_i^\omega$. Also, in this case, we have $-1 \leq \rho_{Rj} \leq 1$, where -1 corresponds to perfect negative correlation and 1 to perfect positive correlation.

When evaluating the insertion of vertex $j$ after the last vertex $i$ of the current route $R$ we use the following function:

$$E_C(R, j) = \alpha \left(\frac{c_{ij}}{\max_{(h,k)\in A} c_{hk}}\right) + (1-\alpha)\rho_{Rj}, \tag{33}$$

where $\alpha$ is a rational number between 0 and 1. The evaluating function is a weighted sum of the normalized travel cost and of the correlation of the insertion as computed in (32). When the insertion of every vertex at the end of the current route is more expensive than creating a new route or when no more vertex can be feasibly inserted, then a new route is created. The insertion procedure terminates when all the vertices are assigned to a route.

*7.1.2. Savings with correlations*

In *Savings with correlations*, we iteratively consider the merging of two routes, if feasible, for all scenarios. In addition, we also take into account the correlation between the two routes, say $R$ and $P$, that we are considering for merging, which is computed as follows:

$$\rho_{RP} = \frac{\sigma_{RP}}{\sigma_R \sigma_P} = \frac{\sum_{\omega\in\Omega} p^\omega \left(\sum_{i\in R} q_i^\omega - \overline{q}_R\right)\left(\sum_{j\in P} q_j^\omega - \overline{q}_P\right)}{\sqrt{\sum_{\omega\in\Omega} p^\omega \left(\sum_{i\in R} q_i^\omega - \overline{q}_R\right)^2}\sqrt{\sum_{\omega\in\Omega} p^\omega \left(\sum_{j\in P} q_j^\omega - \overline{q}_P\right)^2}},$$

where $\overline{q}_X = \sum_{\omega\in\Omega} p^\omega \sum_{i\in X} q_i^\omega$ for route $X$. Thus the evaluating function used when merging two routes results in:

$$E_S(R, P) = \alpha\chi + (1-\alpha)\rho_{RP}, \tag{34}$$

with:

$$\chi = \frac{c_{R(|R|-1),P(1)} - c_{R(|R|-1),0} - c_{0,P(1)}}{\max_{(h,k)\in A} c_{hk} \cdot 2},$$

where $\alpha \in \mathbb{R}$ is a number between 0 and 1, $R(|R|-1)$ is the last vertex of route $R$ before the depot, and $P(1)$ is the first vertex of route $P$ after the depot. In (34) we consider a convex combination of the normalized savings of merging the two routes $R$ and $P$ in this direction and their correlation.

It is worth emphasizing that the initial solution generated by both constructive procedures does not necessarily respect the limit on the maximum number of vehicles even if the algorithms try to minimize the number of routes.

### *7.2. Local search procedure*

The initial solutions are possibly improved by means of a local search procedure based on *Variable Neighborhood Descent* (VND) (Hansen and Mladenović (2001)). Given a predefined neighborhood ordering, the traditional VND works as follows. Initially, it examines the first neighborhood in an exhaustive fashion so as to find the best neighbor. If no improvements were found, then the procedure exhaustively examines the second neighborhood and so on until all neighborhoods are searched without improvements. In case of improvement, then VND restarts the search from the first neighborhood. The following neighborhood structures, whose size are all of the order of $O(n^2)$, were adopted:

- *Relocate*: a vertex is moved from its current position to another one, either in the same route or in a different one. The feasibility of the move is quickly checked in $O(1)$ time for each scenario by means of Properties 3 and 4.

- $\kappa$-*Shift*: at most $\kappa$ consecutive vertices are removed from a route $P$, and then reinserted in another position, either in the same route $P$ or in a different one, say $R$. The feasibility of the removal of vertices from $P$ is checked by evaluating the merging of the two remaining partial routes of $P$ by using Properties 3 and 4. The feasibility of inserting $\kappa$ consecutive vertices in another route $R$ is checked in $O(\kappa)$ operations for each scenario, thus yielding an overall complexity of $O(\kappa|\Omega|)$. This is done by evaluating the $\kappa$ forward and backward load windows of the removed vertices as if they were a single route and by trying to merge it with the first partial route of $R$ via Property 3. If this is feasible then we update the forward load window of the $\kappa$ vertices and we check the merging of the new partial route with the second partial route of $R$ by using the same Property 3. The complexity of

the feasibility check of reinserting $\kappa$ consecutive vertices in route $P$ is slightly different than the previous case. Let $\beta$ be the number of vertices whose position were modified after the reinsertion in $P$. Hence, the number of operations required to check feasibility is $O((\kappa + \beta)|\Omega|)$.

- *Swap*: two vertices are interchanged. This procedures accounts for both inter and intra-route swaps. The feasibility of a swap move is checked in $O(1)$ time via Property 3.

Note that number of routes in the solution is possibly decreased by the *Relocate* and $\kappa$-*Shift* operators.

We have actually implemented a modified version of the traditional VND which is described as follows. At first, the *Relocate* procedure is applied, and when no improvement is obtained the $\kappa$-*Shift* is applied. For the latter, $\kappa$ is initially set as $|V| - 2$ and every time the operator fails to find an improved solution, $\kappa$ is decreased until it reaches value 2. When *2-Shift* does not yield an improvement, the procedure calls once again *Relocate* and $\kappa$-*Shift*. If no improvement at all is obtained after applying *Relocate* and $\kappa$-*Shift* operators, then *Swap* is applied. When *Swap* fails to find an improved solution, the sequence of local searches is repeated until no improvement at all is obtained. We refer to the described implementation as to *NearestVND* and *SavingsVND* in case we make use of the *Nearest neighborhood with Correlation* or *Savings with Correlations* algorithms, respectively, to build the initial solution.

We recall that the objective function of the BRPSD is composed of the travel cost function and the penalty function (also called recourse function, when considering the two stage model). However, in order to speed up the runtime performance while evaluating a move, the algorithm only considers the first function. This is justified by the fact that the penalty cost is not the most relevant in our instances. Nonetheless, when the local search procedure terminates, the algorithm provides the solution obtained to the second stage submodels (see, e.g., Section 4) in order to obtain a set of objective functions, one for each scenario. By summing up all these objective functions multiplied by its probability one can obtain the second component of the objective function of the current solution. This is done only once because it is computationally expensive.

Finally, to speed up our local search procedure, we made use of the so-called *Don't look bits*

strategy (see, e.g., Hoos and Stützle (2004)), that keeps track of possible infeasible moves in a neighborhood and avoids to evaluate those moves more than once. For example, when applying the *Relocate* operator by moving a node in a route and this is infeasible, the algorithm stores that information as a 0-1 value; the value is reset only when the route is modified by other operators.

## 8. Computational experiments

This section presents the computational experiments for the different methodologies developed to solve the BRPSD. The algorithms were run on an Intel Core i3-2100 CPU, with 3.10 GHz and 8.00 GB of RAM, running Windows 7 operating system. CPLEX 12.6 was used as MILP solver and only a single thread was utilized during the testing, which were performed on newly proposed instances derived from real-world data.

### *8.1. Benchmark instances*

We first considered the usage data from a 7-month period concerning the bike sharing system of the city of Reggio Emilia, in Northern Italy. These data were made available by the operator of the system. In addition, we collected data from Capital Bikeshare[1], the bike sharing system located in Washington and Arlington, USA, and also from Divvy[2], located in Chicago, USA. Their websites provide the complete data on the usage of the systems throughout the years. We selected data from the third quarter of 2014, for Capital Bikeshare, and from 30 days in the same period for Divvy. In our instances, each day represents a scenario. We collected the geographic coordinates and the capacity of each station and used historical data to determine the requests of each scenario. It is worth mentioning that some adaptations were necessary because the rebalancing was already applied to the systems and some requests appeared to be very large with respect to the station capacity. In this case, because we had no information on the rebalancing criterion and on the desired level of occupation, we set the request to the maximum capacity of the station. Capital Bikeshare and Divvy are very large bike sharing systems, and at the time of the data collection they had 346 and 296 stations, respectively.

[1]https://www.capitalbikeshare.com

[2]https://www.divvybikes.com

We derived several instances ranging from 20 to 100 vertices by randomly selecting subsets of stations from the complete sets. For each instance, a station was randomly chosen to be the depot.

Benchmark instances are available at `www.or.unimore.it/site/home/online-resources.html`.

### *8.2. Results for the heuristic algorithms based on correlations*

In order to provide good upper bounds for our exact methods we first studied a set of variations of the two proposed heuristics. To perform this evaluation we report the results obtained by *Nearest Neighbor with Correlations* and *Savings with Correlations* and those found by alternative implementations involving several combinations of the developed constructive and local search procedures.

Figure 3 shows the average percentage gap between the best lower bound $LB$ (among those provided by the exact algorithms) and the upper bound $Obj_H$ obtained by the corresponding heuristic, which is computed as follows: $\%gap = 100 \times (Obj_H - LB)/Obj_H$. The experiments were conducted for different values of the parameter $\alpha$ (see (34)) ranging from 0 to 1 with a step of 0.1, and for each configuration, namely: *Nearest Neighbor with Correlations* without local search (*Nearest*), *Nearest+Relocate*, *Nearest+$\kappa$-Shift*, *Nearest+Swap*, *Nearest* with Traditional VND (*NearestTradVND*), *NearestVND*, and the same configurations with Nearest Neighbor substituted by Savings.

From Figure 3, one can notice that *Nearest* does not seem to take advantage of the use of correlations. Indeed, the best average gap is the one provided by $\alpha = 1$, where the correlations are not considered. However, the use of correlations starts showing its importance when the local search procedures are used, this is due to the diversification that the use of correlations provides to the initial solution. This becomes more evident if one focuses on the results of *NearestTradVND* and *NearestVND*, where the average gap obtained when only the correlations were considered ($\alpha = 0$) is the best one. The configuration that yielded the best average gap (17.43%) was *NearestVND* for $\alpha = 0$.

For what concerns the algorithms based on *Savings with Correlations*, the results are similar to those achieved by *Nearest Neighbor with Correlations*, as shown in Figure 3, and the best average gaps are provided by the *SavingsVND* configurations, where the lowest value of 16.37%

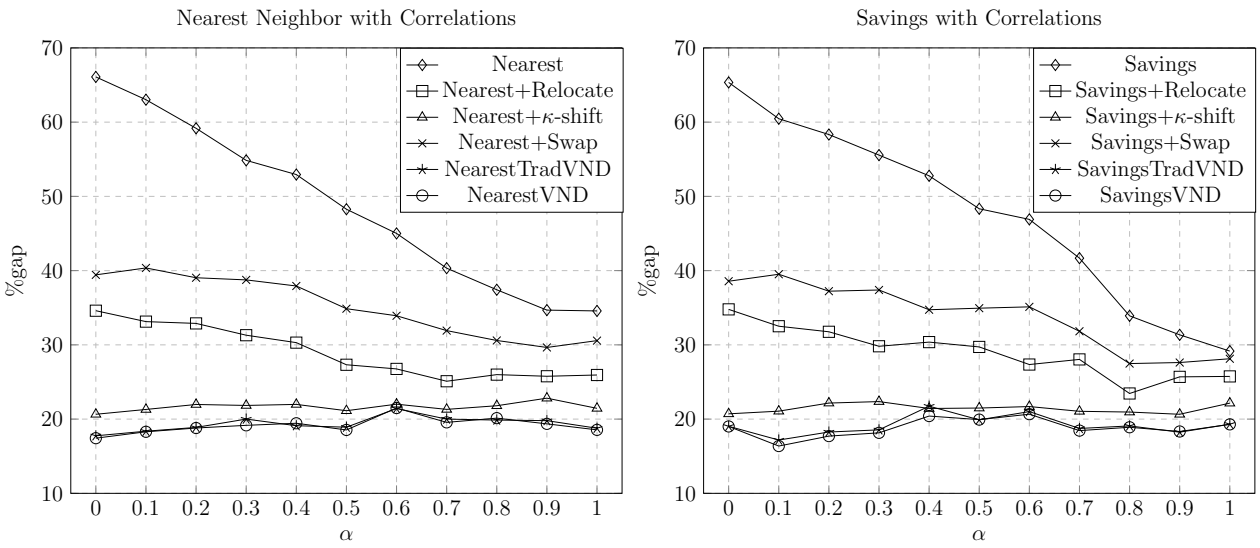


Figure 3: A comparison among different configurations of heuristics with the use of correlations and different values of parameters $\alpha$.

was attained for $\alpha = 0.1$. Therefore, we can verify that the adopted VND improves upon the traditional one.

The average CPU times of the different configurations are very similar and relatively small: the largest one is 4.41 seconds plus about 1 second for the computation of the penalties. Due to the fast CPU times we decided to use a composed algorithm denoted as *InitialUB* running all 22 configurations of *NearestVND* and *SavingsVND*, each one with the 11 studied values of $\alpha = 0, 0.1, \ldots, 0.9, 1$. Table 1 gives the value $InitialUB$ as well as the best lower bounds, $BestLB$, obtained by the exact algorithms (see Section 8.3) along with the percentage gaps computed as $\%gap = 100 \times (InitialUB - BestLB)/InitialUB$, and the CPU time in seconds. Two optimal solutions were found and the gap obtained on two other instances was $< 0.01\%$. The average gap was 11.92%. The CPU time is negligible for small/medium instances and more than 50 seconds only for the largest instance.

### *8.3. Results for the Exact Algorithms*

Table 2 presents the results found by *DEP*, whereas Table 3 shows those obtained by *One-cut L-Shaped* and *Multi-cut L-Shaped*, and Table 4 illustrates the results achieved by *One-cut single stage* and *Multi-cut single stage*. For all exact methods we report, in addition to the name of the instance, the best upper bound ($UB$) and lower bound ($LB$), the percentage gap

Table 1: Best upper bounds obtained by the heuristic algorithms and a comparison with the best known lower bounds.

| Instance | $InitialUB$ | $BestLB$ | $\%gap$ | time |
|---|---|---|---|---|
| Reggio Emilia | 16900.00 | 16900.00 | 0.00 | 0.52 |
| Washington (20)1 | 81195.57 | 81192.12 | $< 0.01$ | 0.52 |
| Washington (20)2 | 79008.23 | 78798.08 | 0.27 | 0.55 |
| Chicago (20)1 | 41066.77 | 40303.87 | 1.86 | 0.19 |
| Chicago (20)2 | 41870.00 | 37592.00 | 10.22 | 0.20 |
| Washington (30)1 | 101980.42 | 101976.19 | $< 0.01$ | 1.26 |
| Washington (30)2 | 118374.40 | 117158.73 | 1.03 | 1.25 |
| Chicago (30)1 | 52401.33 | 45964.00 | 12.28 | 0.52 |
| Chicago (30)2 | 61468.07 | 61468.07 | 0.00 | 0.52 |
| Washington (40)1 | 116498.62 | 99598.41 | 13.95 | 2.92 |
| Washington (40)2 | 229111.18 | 127425.66 | 45.45 | 2.76 |
| Chicago (40)1 | 66029.80 | 65140.93 | 1.35 | 1.26 |
| Chicago (40)2 | 86091.47 | 81766.27 | 5.02 | 1.29 |
| Washington (50)1 | 217415.41 | 130793.99 | 39.14 | 5.83 |
| Washington (50)2 | 136282.02 | 121644.00 | 10.74 | 6.03 |
| Chicago (50) | 92773.60 | 90900.90 | 2.02 | 2.77 |
| Washington (66) | 131319.46 | 130625.10 | 0.53 | 15.80 |
| Chicago (66) | 129969.93 | 109997.00 | 15.37 | 8.43 |
| Washington (80)1 | 139314.97 | 137697.91 | 1.16 | 31.27 |
| Washington (80)2 | 181727.64 | 115461.54 | 36.46 | 29.36 |
| Washington (90) | 231105.37 | 168756.48 | 26.98 | 47.83 |
| Washington (100) | 184979.80 | 158097.00 | 14.53 | 75.00 |
| Average | | | 11.92 | 10.73 |
| St. Dev. | | | 14.01 | 19.08 |
| Maximum | | | 45.45 | 75.00 |
| #opt | | | 2 | |

($\%gap = 100 \cdot (UB - LB)/UB$), the CPU time in seconds (knowing that a time limit of one hour was imposed to the algorithms), and the number of used vehicles (#v) in the best solution. Optimal solutions are highlighted in bold. At the end of the tables we also report the overall average, standard deviation, and maximum percentage gap and time among all instances for which a solution has been found, and the number of optima obtained.

A first analysis of the tables shows that the performance of *DEP* is clearly inferior than the other methods. In particular, for the four largest instances (*Washington (80)1*, *Washington (80)2*, *Washington (90)*, and *Washington (100)*) the 8 GB of memory or our computer were not enough to run the method. The average gap and running times, computed with the remaining instances, definitely show that DEP is not competitive with respect to the other methods.

By observing the results reported in Tables 3 and 4, we notice that the multi-cut-based implementations yielded better results than their respective one-cut counterpart. Although *Multi-cut single stage* obtained inferior gap when compared to *Multi-cut L-Shaped* (average 10.68% against 10.83%, maximum 45.56% against 59.37%), the latter was not dominated by the first, as it was capable of finding better lower bounds in some particular instances, and presents a smaller average CPU time (1847.89 against 2068.21). On the other hand, both methods managed to achieve 11 optimal solutions.

We conclude that multi-cut implementations are the best configurations. In particular, *Multi-cut L-Shaped* should be chosen when quicker convergence is required. If one can wait more CPU time then the *Multi-cut single stage* should be preferred, as it also shows a smaller average standard deviation. A more detailed conclusion on the choice of the two algorithms can be drawn from from the discussion on the values of $\delta$ and $\varepsilon$ that follows.

Concerning the number of vehicles used, we observe that, when it was possible to find the optimal solution, it varied between 1 and 3. Indeed, the use of more than one vehicle can make the penalty cost decrease. For the remaining instances the vehicles used in the best solution found are no more than seven and usually between 1 and 3.

### *8.4. Discussion on the value of $\delta$*

We adopted a value of $\delta = 0.2$ when conducting the experiments reported in the previous sections, which can be considered a reasonable quantity because of the large number of stochastic scenarios considered. We recall that the value of $\delta$ limits the possible slack and surplus

Table 2: Results obtained by *DEP*.

| | DEP | | | | |
|---|---|---|---|---|---|
| Instance | UB | LB | %gap | time | #v |
| Reggio Emilia | 16900.00 | 16900.00 | **0.00** | 6.27 | 1 |
| Washington (20)1 | 81192.12 | 81192.12 | **0.00** | 225.1 | 1 |
| Washington (20)2 | 79008.23 | 76592.57 | 3.06 | 3600 | 1 |
| Chicago (20)1 | 40303.87 | 40303.87 | **0.00** | 742.011 | 3 |
| Chicago (20)2 | 41870.00 | 36025.97 | 13.96 | 3600.00 | 2 |
| Washington (30)1 | 101976.19 | 101976.19 | **0.00** | 474.48 | 1 |
| Washington (30)2 | 117158.73 | 117158.73 | **0.00** | 499.27 | 1 |
| Chicago (30)1 | 51128.00 | 44731.22 | 12.51 | 3600.00 | 1 |
| Chicago (30)2 | 61468.07 | 61468.07 | **0.00** | 419.422 | 1 |
| Washington (40)1 | 116498.62 | 100249.89 | 13.95 | 3600.00 | 1 |
| Washington (40)2 | 229111.18 | 124971.79 | 45.45 | 3600.00 | 3 |
| Chicago (40)1 | 65742.87 | 62712.61 | 4.61 | 3600.00 | 1 |
| Chicago (40)2 | 86091.47 | 77283.10 | 10.23 | 3600.00 | 2 |
| Washington (50)1 | 217415.41 | 95940.94 | 55.87 | 3600.00 | 2 |
| Washington (50)2 | 136282.02 | 117425.96 | 13.84 | 3600.00 | 1 |
| Chicago (50) | 92773.60 | 90231.79 | 2.74 | 3600.00 | 2 |
| Washington (66) | 131319.46 | 104086.97 | 20.74 | 3600.00 | 1 |
| Chicago (66) | 129969.93 | 108273.84 | 16.69 | 3600.00 | 3 |
| Washington (80)1 | 137697.91 | out of mem. | – | – | 1 |
| Washington (80)2 | 181727.64 | out of mem. | – | – | 7 |
| Washington (90) | 231105.37 | out of mem. | – | – | 5 |
| Washington (100) | 184979.8 | out of mem. | – | – | 1 |
| Average | | | 11.87 | 2531.48 | |
| St. Dev. | | | 15.79 | 1560.94 | |
| Maximum | | | 55.87 | 3600.00 | |
| #opt | | | 6 | | |

Table 3: Results obtained by the L-Shaped algorithms.

| | One-cut L-Shaped | | | | | Multi-cut L-Shaped | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | UB | LB | %gap | time | #v | UB | LB | %gap | time | #v |
| Reggio Emilia | 16900.00 | 16900.00 | **0.00** | 1.07 | 1 | 16900.00 | 16900.00 | **0.00** | 0.09 | 1 |
| Washington (20)1 | 81192.12 | 81192.12 | **0.00** | 4.89 | 1 | 81192.12 | 81192.12 | **0.00** | 2.33 | 1 |
| Washington (20)2 | 78824.69 | 78757.08 | 0.09 | 3600.00 | 1 | 78824.69 | 78798.08 | 0.03 | 3600.00 | 1 |
| Chicago (20)1 | 40303.87 | 40303.87 | **0.00** | 106.92 | 3 | 40303.87 | 40303.87 | **0.00** | 54.37 | 3 |
| Chicago (20)2 | 39456.33 | 36750.00 | 6.86 | 3600.00 | 1 | 39456.33 | 37592.00 | 4.73 | 3600.00 | 1 |
| Washington (30)1 | 101976.19 | 101976.19 | **0.00** | 10.32 | 1 | 101976.79 | 101976.19 | **0.00** | 7.49 | 1 |
| Washington (30)2 | 117158.73 | 117158.73 | **0.00** | 3.03 | 1 | 117158.73 | 117158.73 | **0.00** | 2.72 | 1 |
| Chicago (30)1 | 52401.33 | 45964.00 | 12.28 | 3600.00 | 1 | 52401.33 | 45957.00 | 12.30 | 3600.00 | 1 |
| Chicago (30)2 | 61468.07 | 61468.07 | **0.00** | 17.03 | 1 | 61468.07 | 61468.07 | **0.00** | 6.30 | 1 |
| Washington (40)1 | 116498.62 | 97191.00 | 16.57 | 3600.00 | 1 | 116498.62 | 97177.00 | 16.59 | 3600.00 | 1 |
| Washington (40)2 | 229111.18 | 93091.00 | 59.37 | 3600.00 | 3 | 229111.18 | 93086.00 | 59.37 | 3600.00 | 3 |
| Chicago (40)1 | 65140.93 | 65140.93 | **0.00** | 372.49 | 1 | 65140.93 | 65140.93 | **0.00** | 130.39 | 1 |
| Chicago (40)2 | 81766.27 | 80845.00 | 1.13 | 3600.00 | 2 | 81766.27 | 81766.20 | **0.00** | 639.56 | 2 |
| Washington (50)1 | 217415.41 | 128035.00 | 41.11 | 3600.00 | 2 | 217415.41 | 128035.00 | 41.11 | 3600.00 | 2 |
| Washington (50)2 | 136282.02 | 121644.00 | 10.74 | 3600.00 | 1 | 136282.02 | 121644.00 | 10.74 | 3600.00 | 1 |
| Chicago (50) | 90900.90 | 90900.90 | **0.00** | 97.44 | 2 | 90900.90 | 90900.90 | **0.00** | 82.54 | 2 |
| Washington (66) | 130625.10 | 130625.10 | **0.00** | 42.56 | 1 | 130625.10 | 130625.10 | **0.00** | 49.15 | 1 |
| Chicago (66) | 129969.93 | 109980.00 | 15.38 | 3600.00 | 3 | 129969.93 | 109997.00 | 15.37 | 3600.00 | 3 |
| Washington (80)1 | 137697.91 | 137697.91 | **0.00** | 90.547 | 1 | 137697.91 | 137697.91 | **0.00** | 78.627 | 1 |
| Washington (80)2 | 181727.64 | 115388.47 | 36.50 | 3600.00 | 7 | 181727.64 | 115461.54 | 36.46 | 3600.00 | 7 |
| Washington (90) | 231105.37 | 168733.30 | 26.99 | 3600.00 | 5 | 231105.37 | 168756.48 | 26.98 | 3600.00 | 5 |
| Washington (100) | 184979.80 | 158097.00 | 14.53 | 3600.00 | 1 | 184979.80 | 158097.00 | 14.53 | 3600.00 | 1 |
| Average | | | 10.98 | 1997.56 | | | | 10.83 | 1847.89 | |
| St. Dev. | | | 16.42 | 1798.21 | | | | 16.48 | 1797.90 | |
| Maximum | | | 59.37 | 3600.00 | | | | 59.37 | 3600.00 | |
| #opt | | | 10 | | | | | 11 | | |

Table 4: Results by the single stage branch-and-cut algorithms.

| | One-cut single stage | | | | | Multi-cut single stage | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | UB | LB | %gap | time | #v | UB | LB | %gap | time | #v |
| Reggio Emilia | 16900.00 | 16900.00 | **0.00** | 4.59 | 1 | 16900.00 | 16900.00 | **0.00** | 2.72 | 1 |
| Washington (20)1 | 81192.12 | 81192.12 | **0.00** | 20.70 | 1 | 81192.12 | 81192.12 | **0.00** | 8.36 | 1 |
| Washington (20)2 | 79001.85 | 78263.53 | 0.93 | 3600.00 | 1 | 79008.23 | 78334.55 | 0.85 | 3600.00 | 1 |
| Chicago (20)1 | 40303.87 | 40303.87 | **0.00** | 345.45 | 3 | 40303.87 | 40303.87 | **0.00** | 54.50 | 3 |
| Chicago (20)2 | 40847.00 | 35329.22 | 13.51 | 3600.00 | 1 | 39992.67 | 36609.19 | 8.46 | 3600.00 | 1 |
| Washington (30)1 | 101976.19 | 101976.19 | **0.00** | 27.59 | 2 | 101976.19 | 101976.19 | **0.00** | 21.20 | 1 |
| Washington (30)2 | 117158.73 | 117158.73 | **0.00** | 10.72 | 1 | 117158.73 | 117158.73 | **0.00** | 14.71 | 1 |
| Chicago (30)1 | 52401.33 | 45027.16 | 14.07 | 3600.00 | 1 | 52401.33 | 45530.14 | 13.11 | 3600.00 | 2 |
| Chicago (30)2 | 61468.07 | 61468.07 | **0.00** | 35.50 | 1 | 61468.07 | 61468.07 | **0.00** | 10.80 | 1 |
| Washington (40)1 | 116498.62 | 98559.32 | 15.40 | 3600.00 | 1 | 116498.62 | 99416.90 | 14.66 | 3600.00 | 1 |
| Washington (40)2 | 229111.18 | 123782.94 | 45.97 | 3600.00 | 3 | 229111.18 | 124739.04 | 45.56 | 3600.00 | 3 |
| Chicago (40)1 | 66029.80 | 63522.96 | 3.80 | 3600.00 | 1 | 65140.93 | 65140.93 | **0.00** | 1548.06 | 1 |
| Chicago (40)2 | 86091.47 | 76597.63 | 11.03 | 3600.00 | 2 | 81766.27 | 81766.27 | 0.00 | 2497.99 | 2 |
| Washington (50)1 | 217415.41 | 132236.23 | 39.18 | 3600.00 | 2 | 217415.41 | 132314.92 | 39.14 | 3600.00 | 2 |
| Washington (50)2 | 136282.02 | 118213.30 | 13.26 | 3600.00 | 1 | 136282.02 | 118441.00 | 13.09 | 3600.00 | 1 |
| Chicago (50) | 90900.90 | 90900.90 | **0.00** | 451.13 | 2 | 90900.90 | 90900.90 | **0.00** | 406.86 | 2 |
| Washington (66) | 130625.10 | 130625.10 | **0.00** | 489.61 | 1 | 130625.10 | 130625.10 | **0.00** | 380.32 | 1 |
| Chicago (66) | 129969.93 | 105026.02 | 19.19 | 3600.00 | 3 | 129969.93 | 105523.92 | 18.81 | 3600.00 | 3 |
| Washington (80)1 | 137697.91 | 137697.91 | **0.00** | 906.441 | 1 | 137697.91 | 137697.91 | **0.00** | 955.157 | 1 |
| Washington (80)2 | 181727.64 | 111570.63 | 38.61 | 3600.00 | 7 | 181727.64 | 114522.66 | 36.98 | 3600.00 | 7 |
| Washington (90) | 231105.37 | 166110.61 | 28.12 | 3600.00 | 5 | 231105.37 | 166563.46 | 27.93 | 3600.00 | 5 |
| Washington (100) | 184979.80 | 154872.19 | 16.28 | 3600.00 | 1 | 184979.8022 | 154839.4319 | 16.29 | 3600.00 | 1 |
| Average | | | 11.79 | 2231.44 | | | | 10.68 | 2068.21 | |
| St. Dev. | | | 14.53 | 1694.83 | | | | 14.63 | 1666.44 | |
| Maximum | | | 45.97 | 3600.00 | | | | 45.56 | 3600.00 | |
| #opt | | | 9 | | | | | 11 | | |

quantities on requests with respect to the station capacity. In practice, this means that the requests can be violated for each station by up to an amount of 20% of its size for each scenario. Nevertheless, we decided to perform further experiments for different values of $\delta$. For example, for $\delta = 0$, where no slack neither surplus are allowed with respect to requests, and hence the routing solution must be feasible for all scenarios, the value of the objective function increases dramatically. In fact, the average percentage gap between the best lower bound achieved for $\delta = 0$ and the best upper bound attained for $\delta = 0.2$ is around 35%. On the other hand, a larger value such as $\delta = 0.4$, which may be considered relatively soft, appears to be more realistic from the bike sharing provider point of view, and thus more likely to be adopted. Moreover, $\delta = 1$ will let the surplus and slack values limited only by the stations and vehicle capacity, but leave the level of occupation far from the desired one in several stations of the network. All this motivated us to study the behaviour of the two best performing algorithms whit different values of $\delta$, in particular with $\delta = 0, \ldots, 1$ with steps of 0.2. The results are reported in Table 5. The table reports, for each value of $\delta$ and for the two algorithms, the average percentage gap between the upper and lower bounds ($\%gap = 100 \times (UB - LB)/UB$), the average CPU time in seconds (both values computed on all instances), and the number of instances solved to optimality within the time limit of 1 hour. One can see that the average gap is very large for $\delta = 0$. This depends on the fact that we need to respond exactly to each request of every scenario, which can be seen as the worst case scenario: making the problem harder to be solved. This is also linked to the average number of vehicles needed for the instances solved to optimality that, in this case, is 6. For $\delta \geq 0.4$ the number of vehicles used is 1 for all instances. This confirms that parameter $\delta$ is an important element for the flexibility of the routes and that can be a very important factor that drives the changes of costs and solutions.

From Table 5, it can be seen that for the *Multi-cut L-Shaped* the instances become visibly easier when increasing the value of $\delta$, while for the *Multi-cut single stage* this trend is less linear. *Multi-cut L-Shaped* solves 20 instances for $\delta = 0.6, 0.8, 1$ with shorter CPU times and smaller average gaps, thus it appears to be a better option for less constrained problems: it is faster and obtains better solutions. On the other hand, *Multi-cut single stage* appears to be more efficient for harder and more constrained problems, as, e.g., those with $\delta = 0$ or $\delta = 0.2$.

Figure 4 considers the 10 instances that could be solved to optimality for all values of

Table 5: Comparison of multi-cut algorithms when $\delta$ varies.

| | Multi-cut single stage | | | Multi-cut L-Shaped | | |
|---|---|---|---|---|---|---|
| $\delta$ | avg %gap | avg time | #opt | avg %gap | avg time | #opt |
| 0 | 34.06 | 3315.99 | 2 | 47.32 | 3287.54 | 2 |
| 0.2 | 10.68 | 2068.21 | 11 | 10.83 | 1847.89 | 11 |
| 0.4 | 0.85 | 1138.86 | 16 | 0.75 | 872.16 | 17 |
| 0.6 | 0.12 | 704.09 | 20 | 0.03 | 476.00 | 20 |
| 0.8 | 0.11 | 720.43 | 19 | 0.03 | 473.33 | 20 |
| 1 | 0.16 | 710.57 | 19 | 0.03 | 461.31 | 20 |

$\delta > 0$. For each value we report the unsatisfied request averaged on all scenarios (for $\delta = 0$ we know that the number of unsatisfied requests is zero). The results depend on the instance, for *Washington (20)1*, *Washington (30)1*, and *Washington (66)* the values are small and do not vary particularly, while for *Chicago (20)1*, *Chicago (40)1*, *Chicago (40)2*, and *Chicago (50)* the variation is larger and it shows that choosing $\delta = 0.2$ is a good compromise for a low level of average unmet demand. Moreover, one can see that with our method a good solution implies a relatively small number of unmet requests, even if supporting a certain flexibility in the route design.

### *8.5. Discussion on the value of penalty on request violations*

Our stochastic programming problem can be considered a two-criteria problem where the first criterion is the routing cost, while the second is the penalty cost $\varepsilon \cdot c_{\min}$ associated to each unsatisfied request unit (bike). In this section, we report a set of results obtained by varying the value of $\varepsilon$ between 0 and 1, with a step of 0.2 (recall we set $\varepsilon = 0.2$ in all other tests). Value $\varepsilon = 0$ means that no penalty is associated to unsatisfied requests, while $\varepsilon = 1$ imposes each unsatisfied request unit to be penalized by the minimum cost of all arcs. We believe that increasing its value ($\varepsilon > 1$) would not produce interesting results for our problem, given that an unmet request is considered less important than the traveling costs. The value of $\delta$ is fixed to 0.2 as in the original experiments.

Table 6 reports the average percentage gap ($\%gap = 100 \cdot (UB - LB)/UB$), and CPU

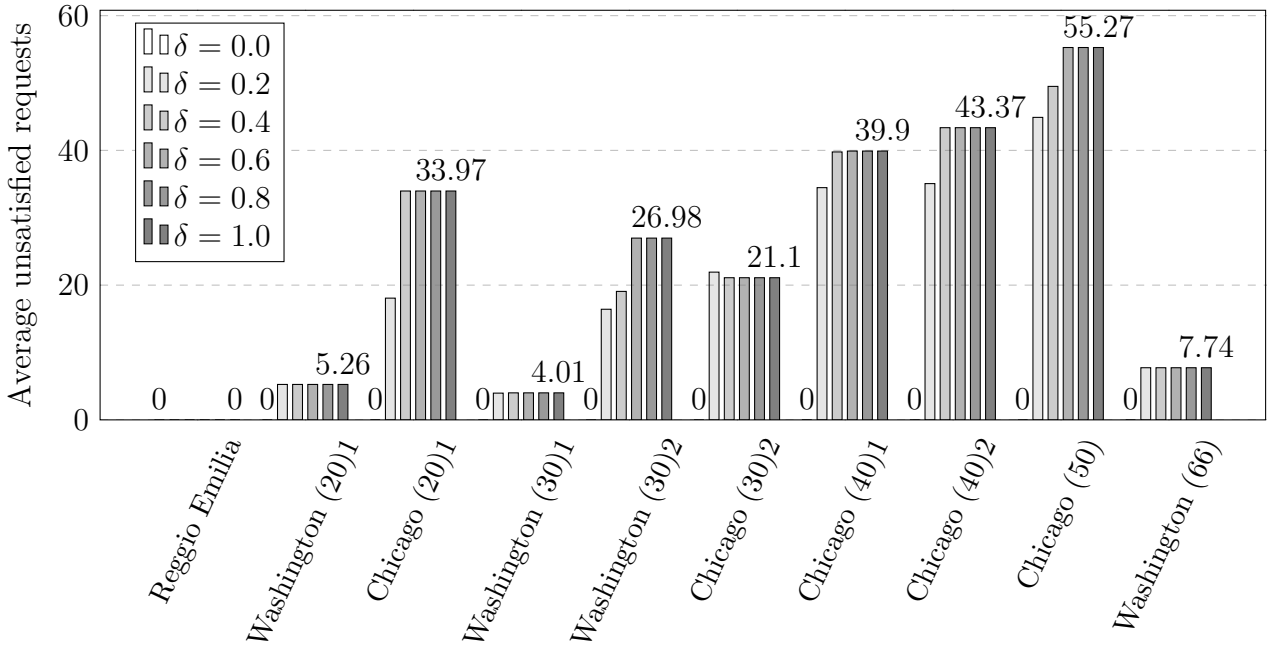


Figure 4: Average unsatisfied requests for different values of $\delta$.

time over all instances, and the number of instances solved to optimality, when $\varepsilon$ increases from 0 to 1 with steps of 0.2 units. We observe that the instances are harder to solve when $\varepsilon$ increases. *Multi-cut single stage* shows better average gaps, but *Multi-cut L-Shaped* is faster and solves to optimality four more instances. This supports, once more, our conclusion on the non domination of one algorithm over the other, and it suggests that *Multi-cut single stage* should be considered when solving more constrained instances such as those with higher $\varepsilon$.

To better understand the impact of the penalty, we considered the instances that could be solved to optimality for all the $\varepsilon$ values and, in Figure 5, we draw a comparison of the average unsatisfied requests for each instance, when $\varepsilon$ changes. The results strongly depend on the instance: for Reggio Emilia no unmet demand was found, while for Washington the solutions do not change or change slightly. For *Chicago (20)1* the average unmet request decreases rapidly. By observing this plot, one can state that the chosen value of $\varepsilon = 0.2$ is reasonable: for most of the instances the average unmet request does not change or changes slightly after $\varepsilon = 0.2$, and for others the value of average unmet request is acceptable for $\varepsilon = 0.2$. On the other hand, this value makes the problem not to hard to solve and let the algorithm to find similar solution is a faster way. The proposed methods show solutions with a relatively

Table 6: Comparison of multi-cut algorithms when $\varepsilon$ varies.

| | Multi-cut single stage | | | Multi-cut L-Shaped | | |
|---|---|---|---|---|---|---|
| $\varepsilon$ | avg %gap | avg time | #opt | avg %gap | avg time | #opt |
| 0.0 | 9.92 | 1858.34 | 12 | 10.15 | 1690.13 | 13 |
| 0.2 | 10.68 | 2068.21 | 11 | 10.83 | 1847.89 | 11 |
| 0.4 | 11.07 | 2181.92 | 10 | 11.59 | 2003.34 | 10 |
| 0.6 | 11.52 | 2268.88 | 9 | 12.13 | 2063.22 | 10 |
| 0.8 | 11.55 | 2358.07 | 8 | 12.76 | 2220.84 | 10 |
| 1.0 | 11.71 | 2360.12 | 8 | 13.19 | 2303.50 | 8 |

small average unmet request; however, this can be decreased by increasing the value of $\varepsilon$ at the expense of a less efficient algorithm.

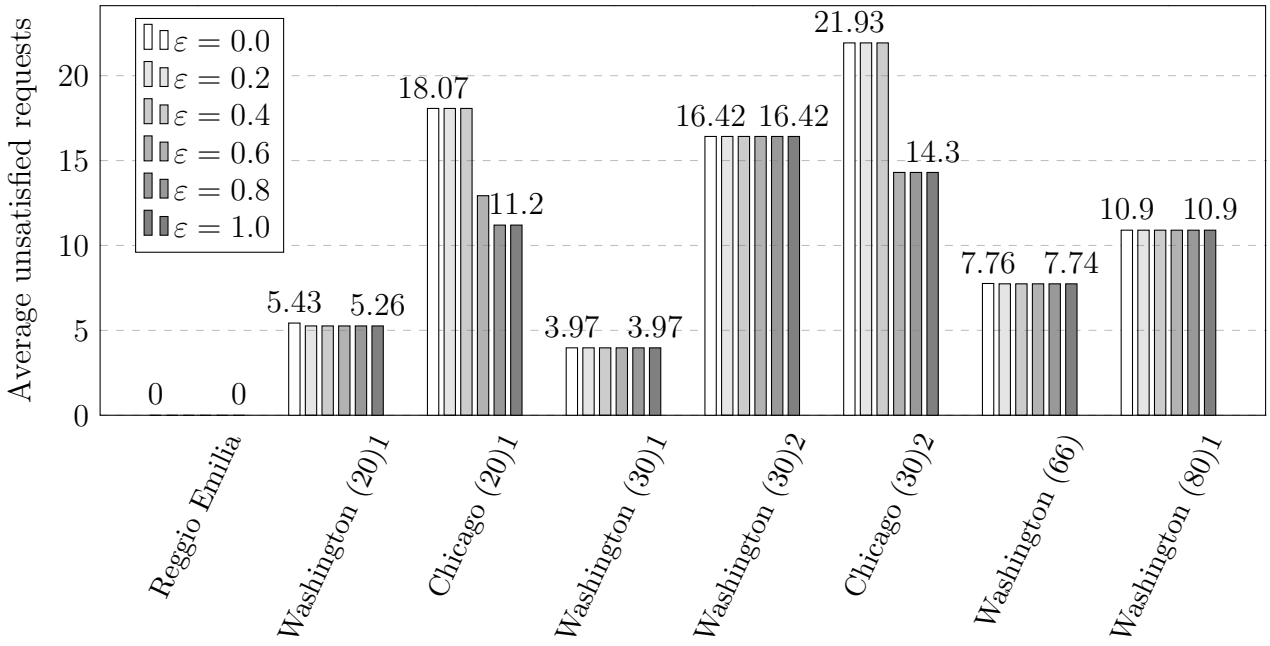


Figure 5: Average unsatisfied requests for different values of penalty $\varepsilon$.

## 9. Concluding Remarks

This paper introduced the *Bike sharing Rebalancing Problem with Stochastic Demands* (BRPSD), a highly challenging problem, given its complexity and potentially large number

of scenarios. To solve it, we proposed five exact procedures based on deterministic equivalent program, L-Shaped methods, and branch-and-cut. We also developed heuristic algorithms that combine novel correlation-based constructive procedures, which take advantage of the information on the requests of the different scenarios, with a VND local search approach. In addition, we presented some properties for the BRPSD that enabled us to implement efficient feasibility checking procedures while evaluating a move. The best upper bound found by the heuristic algorithms was provided as an initial primal bound for the exact methods. The proposed algorithms were tested on newly collected realistic instances and the results that we obtained show their effectiveness. In particular, several optimal solutions were found by the Multi-cut L-Shaped method and also by a Multi-cut single stage branch-and-cut procedure for instances involving up to 80 vertices. The two methods do not seem to dominate each other, but we suggest to use *Multi-cut single stage branch-and-cut* procedure for more constrained problems and *Multi-cut L-Shaped method* for less constrained ones. This is linked to the amount of unmet request that we consider acceptable. With our method we show that we can provide solution with a relatively small amount of unmet request in acceptable CPU times. Moreover, the correlation scheme clearly helped the heuristics to find good quality upper bounds. As for future work one can develop metaheuristic algorithms to improve the quality of the upper bounds, and also tackle other pickup and delivery problems with stochastic components, for which the literature is still very limited. The use of penalties for unmet requests as a recourse function in stochastic vehicle routing problems is also a potential area of investigation. Finally, more complex versions of the problem, such as the one where multiple visits to bike stations are allowed or multiple type of bikes are used could be investigated. Given the larger and larger availability of data, studies using correlations in other routing applications are also an attractive line of future research.

## Acknowledgements

This research was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 305223/2015-1 and 428549/2016-0, by Comissão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), grant PVE A007_2013, and by Ministero dell'Istruzione, dell'Università e della Ricerca, grant PRIN 2015.

## References


Battarra, M., Cordeau, J.-F., Iori, M., 2014. Chapter 6: pickup-and-delivery problems for goods transportation. In: Vehicle Routing: Problems, Methods, and Applications, Second Edition. SIAM, pp. 161–191.

Benchimol, M., Benchimol, P., Chappert, B., De La Taille, A., Laroche, F., Meunier, F., Robinet, L., 2011. Balancing the stations of a self service bike hire system. RAIRO-Operations Research 45 (1), 37–61.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., Laporte, G., 2007. Static pickup and delivery problems: a classification scheme and survey. Top 15 (1), 1–31.

Bertsimas, D. J., Simchi-Levi, D., 1996. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. Operations Research 44 (2), 286–304.

Birge, J. R., Louveaux, F., 2011. Introduction to stochastic programming. Springer Science & Business Media.

Bruck, B. P., Cruz, F., Iori, M., Subramanian, A., 2017. The static bike sharing rebalancing problem with forbidden temporary operations. Technical report.

Bulhões, T., Subramanian, A., Erdoğan, G., Laporte, G., 2018. The static bike relocation problem with multiple vehicles and visits. European Journal of Operational Research 264 (2), 508–523.

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. Cirrelt Montreal.

Crowder, H., Padberg, M., 1980. Solving large-scale symmetric travelling salesman problems to optimality. Management Science 26, 495–509.

Cruz, F., Subramanian, A., Bruck, B. P., Iori, M., 2017. A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. Computers & Operations Research 79, 19–33.

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 & Operations Research 71, 149–162.

Di Gaspero, L., Rendl, A., Urli, T., 2013. A hybrid ACO+CP for balancing bicycle sharing systems. In: International Workshop on Hybrid Metaheuristics. Springer, pp. 198–212.

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

Erdoğan, G., Laporte, G., Wolfler Calvo, R., 2014. The static bicycle relocation problem with demand intervals. European Journal of Operational Research 238 (2), 451–457.

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.

Gendreau, M., Jabali, O., Rei, W., 2014. Vehicle Routing: problems methods and applications. SIAM, Philadelphia, Ch. Stochastic Vehicle Routing Problems, pp. 213–239.

Gendreau, M., Jabali, O., Rei, W., 2016. 50th anniversary invited articlefuture research directions in stochastic vehicle routing. Transportation Science 50 (4), 1163–1173.

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

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 & Operations Research 36 (5), 1639–1645.

Hernández-Pérez, H., Salazar-González, J.-J., 2004. 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., 2007. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. Networks 50 (4), 258–272.

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.

Hoos, H. H., Stützle, T., 2004. Stochastic local search: Foundations and applications. Elsevier.

King, A. J., Wallace, S. W., 2012. Modeling with stochastic programming. Springer Science & Business Media.

Laporte, G., Louveaux, F., Van Hamme, L., 2002. An integer l-shaped algorithm for the capacitated vehicle routing problem with stochastic demands. Operations Research 50 (3), 415–423.

Li, Y., Szeto, W., Long, J., Shui, C., 2016. A multiple type bike repositioning problem. Transportation Research Part B: Methodological 90, 263–278.

Louveaux, F., Salazar-González, J.-J., 2009. On the one-commodity pickup-and-delivery traveling salesman problem with stochastic demands. Mathematical Programming 119 (1), 169–194.

Louveaux, F., Salazar-González, J.-J., 2014. Solving the single vehicle routing problem with variable capacity. Transportation Science 50 (2), 708–719.

Meddin, R., DeMaio, P., 2018. The bike-sharing world map. URL http://www. metrobike. net.

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.

Oyola, J., Arntzen, H., Woodruff, D. L., 2016. The stochastic vehicle routing problem, a literature review. Technical Report.

Pearson, K., 1920. Notes on the history of correlation. Biometrika 13 (1), 25–45.

Rainer-Harbach, M., Papazek, P., Raidl, G. R., Hu, B., Kloimüllner, C., 2015. Pilot, grasp, and vns approaches for the static balancing of bicycle sharing systems. Journal of Global Optimization 63 (3), 597–629.

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.

Regue, R., Recker, W., 2014. Proactive vehicle routing with inferred demand to solve the bikes-haring rebalancing problem. Transportation Research Part E: Logistics and Transportation Review 72, 192–209.

Saharidis, G. K. D., Fragkogios, A., Zygouri, E., 2014. A multi-periodic optimization modeling approach for the establishment of a bike sharing network: A case study of the city of athens. In: Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. 2.

Salazar-González, J.-J., Santos-Hernández, B., 2015. The split-demand one-commodity pickup-and-delivery travelling salesman problem. Transportation Research Part B: Methodological 75, 58–73.

Schuijbroek, J., Hampshire, R. C., Van Hoeve, W.-J., 2017. Inventory rebalancing and vehicle routing in bike sharing systems. European Journal of Operational Research 257 (3), 992–1004.

Shui, C., Szeto, W., 2017. Dynamic green bike repositioning problem–a hybrid rolling horizon artificial bee colony algorithm approach. Transportation Research Part D: Transport and Environment.

Stewart, W. R., Golden, B. L., 1983. Stochastic vehicle routing: A comprehensive approach. European Journal of Operational Research 14 (4), 371–385.

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.

Szeto, W., Shui, C., 2018. Exact loading and unloading strategies for the static multi-vehicle bike repositioning problem. Transportation Research Part B: Methodological 109, 176–211.

Toth, P., Vigo, D., 2014. Vehicle routing: problems, methods, and applications. SIAM.

Wang, F., Lim, A., Xu, Z., 2006. The one-commodity pickup and delivery travelling salesman problem on a path or a tree. Networks 48 (1), 24–35.

Wang, I.-L., Wang, C.-W., 2013. Analyzing bike repositioning strategies based on simulations for public bike sharing systems: simulating bike repositioning strategies for bike sharing systems. In: Advanced Applied Informatics (IIAIAAI), 2013 IIAI International Conference on. IEEE, pp. 306–311.

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.