A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem

Sin C. Ho[a] and W. Y. Szeto[b,c]

[a]Department of Economics and Business Economics
Aarhus University, Denmark
sinho@econ.au.dk

[b]The University of Hong Kong Shenzhen Institute of Research and Innovation
Shenzhen, China

[c]Department of Civil Engineering
The University of Hong Kong, Hong Kong
ceszeto@hku.hk

Abstract

This paper addresses the multi-vehicle bike-repositioning problem, a pick-up and delivery vehicle routing problem that arises in connection with bike-sharing systems. Bike-sharing is a green transportation mode that makes it possible for people to use shared bikes for travel. Bikes are retrieved and parked at any of the stations within the bike-sharing network. One major challenge is that the demand for and supply of bikes are not always matched. Hence, vehicles are used to pick up bikes from surplus stations and transport them to deficit stations to satisfy a particular service level. This operation is called a bike-repositioning problem. In this paper, we propose a hybrid large neighborhood search for solving the problem. Several removal and insertion operators are proposed to diversify and intensify the search. A simple tabu search is further applied to the most promising solutions. The heuristic is evaluated on three sets of instances with up to 518 stations and five vehicles. The results of computational experiments indicate that the heuristic outperforms both CPLEX and the math heuristic proposed by Forma et al. (2015) [Transportation Research Part B 71: 230-247]. The average improvement of our heuristic over the math heuristic is 1.06%, and it requires only a small fraction of the computation time.

Keywords: Bike-sharing; Bike-repositioning; Pick-up and delivery routing; Large neighborhood search; Tabu search

1 Introduction

Bikes constitute a green and healthy mode of transportation, and have thus drawn increased attention in recent years. Research topics include bike trip estimation (de Chardon and Caruso, 2015), bike network design (Lin and Yang, 2011; Lin et al., 2013; Chow and Sayarshad, 2014), bike network flow analysis (Kitthamkesorn et al., 2016), bike service level analysis (Raviv and Kolka, 2013), bike safety (Lawson et al., 2013), bike redistribution strategies (Nair and Miller-Hooks, 2011), and bike repositioning. In bike repositioning, vehicles are deployed to pick up and transport bikes from stations with an excess of bikes to stations with an insufficient number. Table 1 summarizes the literature on bike-repositioning problems according to operation type, number of repositioning vehicles used, and problem objectives.

In terms of operation type, the literature can be roughly classified into two categories: static and dynamic. Static repositioning problems consider night-time operations and scenarios in which demand is low or the system is closed, meaning that the change in demand is negligible. Dynamic repositioning problems mainly consider daytime operations and scenarios that take real-time system usage into account. As shown in Table 1, most studies focus on static repositioning problems because such problems are already difficult to analyze and solve without introducing further complexities. Ho and Szeto (2014) pointed out that static repositioning problems are NP-hard, and are more difficult to solve than classical routing problems because of the presence of pick-up and drop-off quantities as decision variables. An understanding of static repositioning problems and the algorithms developed for them is useful in addressing more difficult dynamic repositioning problems.

Table 1: Summary of the bike-repositioning problem literature

Reference	Type	No. of vehicles	Objective
Benchimol et al. (2011)	Static	1	Minimize total travel cost
Caggiani and Ottomanelli (2012)	Dynamic	> 1	Minimize relocation and lost user cost
Contardo et al. (2012)	Dynamic	> 1	Minimize total unmet demand
Lin and Chou (2012)	Static	> 1	Minimize total travel time or distance
Chemla et al. (2013)	Static	1	Minimize total travel distance
Di Gaspero et al. (2013a)	Static	> 1	Minimize the weighted sum of total travel time and total absolute deviation from the target number of bikes
Nair et al. (2013)	Static	1	Minimize total redistribution cost
Raviv et al. (2013)	Static	> 1	Minimize the weighted sum of total travel time and penalty cost
Schuijbroek et al. (2013)	Static	> 1	Minimize maximum tour length
Erdoğan et al. (2014)	Static	1	Minimize travel and handling costs
Ho and Szeto (2014)	Static	1	Minimize total penalty cost
Kloimüllner et al. (2014)	Dynamic	≥ 1	Minimize the weighted sum of unfulfilled demand, absolute deviation from the target fill level, total number of loading instructions, and total drive time
Forma et al. (2015)	Static	> 1	Minimize the weighted sum of total travel time and penalty cost
Rainer-Harbach et al. (2015)	Static	≥ 1	Minimize the weighted sum of the total absolute deviation from the target number of bikes, total number of loading/unloading activities, and overall travel time required for all routes
Szeto et al. (2016)	Static	1	Minimize the weighted sum of unmet customer demand and operational time on the vehicle route

The objectives considered in the literature vary. As shown in Table 1, both single and weighted sum objectives are considered. The objectives are formed by either a single measure of effectiveness (e.g., total unmet demand) or a weighted combination of measures of effectiveness (e.g., the weighted sum of unfulfilled demand, the absolute deviation from the target fill level, the total

number of loading instructions, and total drive time). Moreover, travel time or distance, user dissatisfaction, and penalty cost are commonly used as sole or partial components in the objective function. The choice of objectives should be determined by the application of bike-sharing operations. The operator's concern normally governs the choice of objective. Meanwhile, some objectives are more general than others. For example, minimizing total penalty cost is more general than minimizing total user dissatisfaction or the sum of the deviations from the target number of bikes in each station because we can choose a penalty function that assigns a value of zero to the level equal to or greater than the demand level and a very large number to other levels to replicate the effect of minimizing total user dissatisfaction. Similarly, we can select a penalty function that assigns a value to a level equal to the absolute difference between that level and the target level to replicate the effect of minimizing the sum of deviations.

The literature can also be classified according to the number of vehicles employed. In terms of formulation, multiple-vehicle repositioning problems are straightforward extensions of single-vehicle problems. However, it is more realistic to consider multiple-vehicle repositioning problems. Some studies that consider multiple vehicles (Alvarez-Valdes et al., 2016) allow each station to be visited by multiple vehicles more than once, whereas others (Dell'Amico et al., 2014) allow each station to be visited only by exactly one vehicle. The main challenge in addressing multi-vehicle than single-vehicle repositioning problems is developing efficient solution methods to handle the larger solution space arising from the presence of more vehicles and the possibility of multiple visits to a station. Direct applications of the solution techniques for the single-vehicle case cannot search the solution space efficiently.

Exact methods such as branch-and-cut algorithms (see Dell'Amico et al., 2014; Erdoğan et al., 2014; Erdoğan et al., 2015) have been used to solve repositioning problems. However, such methods are intractable for large, realistic repositioning problems. The literature (e.g., Raviv et al., 2013; Ho and Szeto, 2014) has also illustrated this point via numerical experiments. Therefore, most studies to date have focused on developing inexact methods to obtain good solutions using small computing time. A brief summary of inexact solution methods follows.

Approximation method:
9.5-approximation algorithm (Benchimol et al., 2011)
Heuristics or metaheuristics:
Ant colony and constraint programming (Di Gaspero et al., 2013b)
Cluster-first route-second (Schuijbroek et al., 2013)
Iterated tabu search (Ho and Szeto, 2014)
GRASP with path relinking (Papazek et al., 2014)
GRASP with VND (Kloimüllner et al., 2014; Rainer-Harbach et al., 2015)
VNS (Kloimüllner et al., 2014; Rainer-Harbach et al., 2015)
Chemical reaction optimization (Szeto et al., 2016)
Hybrid methods (of exact method and heuristic):
Branch-and-cut method with tabu search (Chemla et al., 2013)
3-step math heuristic (Forma et al., 2015)

GRASP: Greedy randomized adaptive search procedure; VND: Variable neighborhood descent; VNS: Variable neighborhood search

It can be seen from the foregoing summary that few hybrid methods have been developed and that recent heuristics such as large neighborhood search (LNS) have not been applied to bike-repositioning problems. LNS is a metaheuristic in which, at each iteration, a part of the solution

is destroyed by a removal operator, and the solution is repaired by an insertion operator. LNS differs from variable neighborhood search (VNS). According to Mladenović and Hansen (1997), VNS is a metaheuristic that systematically performs the neighborhood change procedure, both in descent to local minima and in escape from the valleys that contain them.

Most of the preceding inexact methods do not consider problem properties and station characteristics. For example, in reality, not all stations need to be visited by repositioning vehicles for several reasons. First, the stations where the demand for bikes equals the supply are not required to provide extra bikes when the objective is to minimize unmet demand. The bikes at these stations should also not be taken away. Hence, it is unnecessary to visit these "balanced" stations. Second, some stations may not be reached by vehicles owing to short operational time. Third, it is not necessary to visit all stations in an optimal solution because of shortage of repositioning resources (e.g., limited repositioning time) and costs (e.g., the fact that it is not worthy to let a vehicle make a very long trip to save tiny penalty cost). Finally, there may be an insufficient total supply from pick-up stations for drop-off stations. In such a case, even if trucks visited all drop-off stations, the total demand from those stations could not be satisfied. Hence, it is unwise to visit all drop-off stations.

To the best of our knowledge, only Ting and Liao (2013), Ho and Szeto (2014; 2016), and Szeto et al. (2016) have considered station characteristics to narrow the solution search space and develop efficient heuristics to solve their problems. These studies classified stations into pick-up and drop-off stations and made use of station characteristics in problem-solving. Ting and Liao (2013) and Ho and Szeto (2016) considered the total travel time of a vehicle in the objective function, and the number of bikes required by each drop-off station was explicitly given. Ho and Szeto (2016) developed a GRASP with path relinking to solve the problem examined by Ting and Liao (2013), and showed it to produce better average results than the memetic algorithm used by Ting and Liao (2013) in less computing time. However, they did not consider a penalty cost or loading/unloading quantities. The penalty cost of not satisfying one unit of demand may vary from one station to another. In an area with high (low) bike station density or many (few) transportation alternatives, the penalty cost may be low (high) because users can (cannot) easily walk to nearby bike stations or take other modes of transportation. It is more reasonable to consider such a penalty cost than user demand alone for bike services offered by governments. Ho and Szeto (2014) refined the arc-based formulation of the static repositioning problem in Raviv et al. (2013) to minimize the total penalty cost. They also developed an efficient iterated tabu search for large repositioning network applications. However, they consider the single-vehicle case alone and do not consider total travel time as a secondary objective. Szeto et al. (2016) also studied the single-vehicle static bike-repositioning problem but did not consider the penalty cost in the objective function. The objective of repositioning is to minimize the weighted sum of unmet customer demand and operational time on the vehicle route. The problem was solved by the chemical reaction optimization.

This paper develops a hybrid large neighborhood search (H-LNS) to solve a problem similar to the multi-vehicle static repositioning problem in Raviv et al. (2013) and Forma et al. (2015). This problem is the night-time operation problem of using more than one vehicle to pick up bikes from locations with excess bikes and transport them to stations with insufficient bikes to address the bike imbalance issue. It determines the sequence of stations for each repositioning vehicle to visit, the pick-up quantity from each station with excess bikes for each vehicle, and the drop-off quantity to each station with insufficient bikes from each vehicle. Station characteristics are incorporated into the formulation and used in designing the algorithm. The hybrid algorithm incorporates tabu search to improve the local search ability of LNS. To demonstrate the efficiency and accuracy of our method, we set up various test scenarios and compared the results with those obtained from IBM ILOG CPLEX and the 3-step math heuristic proposed by Forma

et al. (2015). The results show our algorithm to obtain better solutions than the 3-step math heuristic in much less computing time. Computational tests were also performed to confirm that H-LNS outperforms LNS (without incorporating tabu search). The results are presented herein to illustrate the contribution of each removal and insertion operator to solution accuracy.

This paper differs from Ho and Szeto (2014) in three main respects: 1) it studies a multiple- rather than single-vehicle repositioning problem, 2) it considers travel time in the objective function, and 3) it develops a solution algorithm based on a different solution approach as a backbone algorithm and more insertion and removal operators. Compared with Ho and Szeto (2016), this paper also considers a more complicated routing problem, with pick-up and drop-off quantities as the decision variables and multiple vehicles with split pick-ups and deliveries. This paper also differs from Szeto et al. (2016) in three aspects: 1) it studies a multiple-vehicle bike repositioning problem, 2) it considers the penalty cost in the objective function, and 3) it develops a different solution method.

The paper's contributions include the following.

1. This paper develops an efficient and effective hybrid heuristic to solve large real-life instances of multiple-vehicle bike repositioning problems.
2. To the best of our knowledge, this is the first application of large neighborhood search to solve multiple-vehicle bike repositioning problems with great success.

The rest of the paper is organized as follows. Section 2 presents the mathematical formulation. Section 3 depicts the hybrid heuristic. Section 4 presents the test cases and discusses the results. Section 5 concludes the paper.

2 Mathematical formulation

This section presents the arc-indexed formulation of Raviv et al. (2013) for the sake of completeness and the revised model with additional constraints to explicitly consider the characteristics of the pick-up and drop-off stations. A pick-up station is a station at which the initial number of bikes is greater than the optimal number (i.e., the level at which the station's penalty function attains its minimum), whereas a drop-off station is a station at which the initial number of bikes is smaller than the optimal number.

The repositioning problem considers a set of stations and a depot. Each station is characterized by its capacity, initial bike inventory, and a penalty function. The penalty function represents the expected shortage of bikes and lockers incurred by station users on the next day as a function of the station's inventory after repositioning is carried out (Forma et al., 2015). The depot is assumed to have a very large capacity and no demand.

Multiple repositioning vehicles with limited capacity collect bikes from pick-up stations and transport them to and unload them at drop-off stations. These vehicles start from and end up at the depot and operate within a given time constraint. They are allowed to return to the depot to load or unload bikes in the middle of the operation. The objective is to determine the route of each vehicle and the quantities of bikes loaded and unloaded at each station visited to minimize the weighted sum of the penalty cost at each station and the total travel time.

2.1 Notations

The following notations are used throughout the paper.

Sets

$\mathcal{N}$	Set of stations
$\mathcal{N}_0$	Set of nodes, including the stations and depot
$\mathcal{P}$	Set of pick-up stations
$\mathcal{D}$	Set of drop-off stations
$\mathcal{V}$	Set of vehicles

Indices

i, j	Indices of nodes
v	Index of vehicle (or route)
$\bar{u}$	Index of inventory level

Parameters

α	Weight of total travel time relative to total penalty cost
s_i^0	Initial inventory at station i
c_i	Capacity of station i
k	Vehicle capacity
T	Repositioning time
L	Time required to load a bike from a station onto a vehicle
U	Time required to unload a bike from a vehicle to a station
M	A very large number
a_i, b_i	Parameters associated with the penalty function for station i
t_{ij}	Travel time from station i to station j

Decision variables

x_{ijv}	Binary variable that equals one if vehicle v travels directly from node i to node j, and zero otherwise
y_{ijv}	Number of bikes on vehicle v when it travels directly from node i to node j
q_{iv}	Auxiliary variable associated with node i used for the sub-tour elimination constraints
y_{iv}^P	Number of bikes loaded onto vehicle v at node i
y_{iv}^D	Number of bikes unloaded from vehicle v at node i
s_i	Inventory level at station i at the end of the repositioning operation

Function

$f_i(s_i)$	A convex penalty function for station i defined over s_i

2.2 Arc-indexed formulation

The basic arc-indexed formulation in Raviv et al. (2013) is given by

penalty function?

objective function

$$\min \sum_{i \in \mathcal{N}} f_i(s_i) + \alpha \sum_{i \in \mathcal{N}_0} \sum_{j \in \mathcal{N}_0, j \neq i} \sum_{v \in \mathcal{V}} t_{ij} x_{ijv} \tag{1}$$

final number of bike

$$\text{s. t. } s_i = s_i^0 - \sum_{v \in \mathcal{V}} (y_{iv}^P - y_{iv}^D) \qquad \forall i \in \mathcal{N}_0 \tag{2}$$

$$y_{iv}^P - y_{iv}^D = \sum_{j \in \mathcal{N}_0, j \neq i} y_{ijv} - \sum_{j \in \mathcal{N}_0, j \neq i} y_{jiv} \qquad \forall i \in \mathcal{N}_0, \forall v \in \mathcal{V} \tag{3}$$

amount loaded cannot larger than vehicle capacity

$$y_{ijv} \leq k x_{ijv} \qquad \forall i, j \in \mathcal{N}_0, i \neq j, \forall v \in \mathcal{V} \tag{4}$$

must leave

$$\sum_{j \in \mathcal{N}_0, j \neq i} x_{ijv} = \sum_{j \in \mathcal{N}_0, j \neq i} x_{jiv} \qquad \forall i \in \mathcal{N}_0, \forall v \in \mathcal{V} \tag{5}$$

at most visit one time

$$\sum_{j \in \mathcal{N}_0, j \neq i} x_{ijv} \leq 1 \qquad \forall i \in \mathcal{N}, \forall v \in \mathcal{V} \tag{6}$$

at most pick up

$$\sum_{v \in \mathcal{V}} y^P_{iv} \leq s^0_i \qquad \forall i \in \mathcal{N}_0 \tag{7}$$

at most drop off

$$\sum_{v \in \mathcal{V}} y^D_{iv} \leq c_i - s^0_i \qquad \forall i \in \mathcal{N}_0 \tag{8}$$

all delivered, no remain bike at vehicle

$$\sum_{i \in \mathcal{N}_0} (y^P_{iv} - y^D_{iv}) = 0 \qquad \forall v \in \mathcal{V} \tag{9}$$

$$\sum_{i \in \mathcal{N}} (L y^P_{iv} + U y^D_{iv}) + \sum_{i \in \mathcal{N}} (L y_{0iv} + U y_{i0v}) + \sum_{i,j \in \mathcal{N}_0 : i \neq j} t_{ij} x_{ijv} \leq T \qquad \forall v \in \mathcal{V} \tag{10}$$

Travelling Salesman Problem

$$q_{jv} \geq q_{iv} + 1 - M(1 - x_{ijv}) \qquad \forall i \in \mathcal{N}_0, j \in \mathcal{N}, i \neq j, \forall v \in \mathcal{V} \tag{11}$$

salesman is required to visit each of n cities, indexed by 1, … , n. He leaves from a "base city" ndexed by 0, visits each of the n other cities exactly once, and returns to city 0. Minimize total sub-tour: tour that not start and end at depot distance

$$x_{ijv} \in \{0, 1\} \qquad \forall i, j \in \mathcal{N}_0 : i \neq j, \forall v \in \mathcal{V} \tag{12}$$

$$y^P_{iv} \geq 0, y^D_{iv} \geq 0, \text{integer} \qquad \forall i \in \mathcal{N}_0, \forall v \in \mathcal{V} \tag{13}$$

$$y_{ijv} \geq 0 \qquad \forall i, j \in \mathcal{N}_0 : i \neq j, \forall v \in \mathcal{V} \tag{14}$$

$$s_i \geq 0 \qquad \forall i \in \mathcal{N}_0 \tag{15}$$

$$q_{iv} \geq 0 \qquad \forall i \in \mathcal{N}_0 \tag{16}$$

The objective function (1) is defined as the sum of the penalty cost incurred at each station and the weighted total travel time. Equation (2) defines the final number of bikes at each node as the sum of the initial number of bikes at that node plus the total number of bikes loaded onto and unloaded from each vehicle. Equation (3) requires that the number of bikes loaded onto or unloaded from a vehicle at a given node equals the difference between the vehicle load before and after the node visit. Constraint (4) ensures that the load on each vehicle cannot be greater than the vehicle capacity. Equation (5) ensures that if a vehicle visits a station, it must leave that station. Constraint (6) ensures that each vehicle can visit a station at most once. Constraints (7) and (8) require that the total pick-up and drop-off quantities at each node are not larger than the number of bikes available at and the remaining capacity of that node, respectively. Constraint (9) ensures that all bikes picked up by each vehicle are eventually delivered. Constraint (10) ensures that the sum of the loading, unloading, and travel times of each vehicle does not exceed the repositioning time available. Constraint (11) is the sub-tour elimination constraint (see Miller et al., 1960), and constraints (12)-(16) are the domain constraints.

Raviv et al. (2013) later linearized the objective function as

$$\min \sum_{i \in \mathcal{N}} g_i + \alpha \sum_{i \in \mathcal{N}_0} \sum_{j \in \mathcal{N}_0, j \neq i} \sum_{v \in \mathcal{V}} t_{ij} x_{ijv}, \tag{17}$$

and added the following linear constraints to the formulation.

$$g_i \geq a_{i\bar{u}} + b_{i\bar{u}} s_i \qquad \forall i \in \mathcal{N}, \bar{u} = 0, \ldots, c_i - 1, \tag{18}$$

at is fi here? Penalty function? gi: new penalty function?

where g_i is the penalty incurred at station i,

What is this linear formulation represent?

$$b_{i\bar{u}} \equiv f_i(\bar{u}+1) - f_i(\bar{u})$$

Penalty function min -> optimal?

$$a_{i\bar{u}} \equiv f_i(\bar{u}) - b_{i\bar{u}} \cdot \bar{u}.$$

To speed up computation, they also added the following.

departs from the depot at least 1

$$\sum_{j \in \mathcal{N}} x_{0jv} \geq 1 \qquad \forall v \in \mathcal{V} \tag{19}$$

Former part: To tighten the loading quantities, cannot larger than vehicle capacity

$$y^P_{iv} \leq \min(s^0_i, k) \sum_{j \in \mathcal{N}_0} x_{ijv} \qquad \forall i \in \mathcal{N}, \forall v \in \mathcal{V} \tag{20}$$

Later part: Why multiple summation? Multiple visit time?

$$y^D_{iv} \leq \min(c_i - s^0_i, k) \sum_{j \in \mathcal{N}_0} x_{ijv} \qquad \forall i \in \mathcal{N}, \forall v \in \mathcal{V} \tag{21}$$

number of loading/unloading activity >= visit time

$$y^P_{iv} + y^D_{iv} \geq \sum_{j \in \mathcal{N}_0} x_{ijv} \qquad \forall i \in \mathcal{N}, \forall v \in \mathcal{V} \tag{22}$$

$$\sum_{j \in \mathcal{N}} j \cdot x_{0jv} \leq \sum_{j \in \mathcal{N}} j \cdot x_{0,j,v+1} \qquad \forall v \in \mathcal{V} \tag{23}$$

Symmetry-breaking constraint.

The vehicles cannot have the same route? why j times x

Constraint (19) ensures that each vehicle departs from the depot at least once. Constraints (20)-(21) further tighten the solution space for the loading and unloading quantities of each vehicle at a station, respectively, by including vehicle capacity and conditioning the quantities only for cases in which the corresponding vehicle visits the station. Constraint (22) tightens the solution space by ensuring that each vehicle that enters a station must engage in a loading or unloading activity, and constraint (23) is the the symmetry-breaking constraint.

2.3 Revised model

Our revised model is as follows.

min (17)
s. t. (2)-(16) and (18)-(23) and the following station characteristic constraints.

drop off station dun pick-up

$$y^P_{iv} = 0 \qquad \forall i \in \mathcal{D}, \forall v \in \mathcal{V} \tag{24}$$

pick-up station dun drop-off

$$y^D_{iv} = 0 \qquad \forall i \in \mathcal{P}, \forall v \in \mathcal{V} \tag{25}$$

It is clear from the definitions of the pick-up and drop-off stations that $\mathcal{P} = \{i \in \mathcal{N} | s^0_i > s^I_i\}$ and $\mathcal{D} = \{i \in \mathcal{N} | s^0_i < s^I_i\}$, where $s^I_i = \arg\min_{s_i}\{f_i(s_i)\}$ (i.e., the optimal number of bikes at node i). These stations and the optimal quantities are known before the operation starts.

It is also clear that, in principle, the revised model's optimal objective value should not be smaller than the basic model of Raviv et al. (2013) because our solution space is smaller. However, adding station characteristic constraints (24)-(25) into their model can benefit the development of efficient and effective heuristics to solve our multi-vehicle static repositioning problem. We can make use of the characteristics of each station (e.g., a pick-up, drop-off, or balanced station) in the revised model to develop an efficient and effective heuristic to solve the problem under study. As shown in Section 4, even though we solve a more restricted problem, our solution method obtains a better solution faster than the math heuristic of Forma et al. (2015), which was developed to solve the model in Section 2.2.

3 The algorithm

In this section, we present a hybrid large neighborhood search for solving the multi-vehicle bike-repositioning problem. LNS is a metaheuristic in which, at each iteration, a part of the solution is destroyed by an operator and the solution is repaired by another operator. LNS was first applied by Shaw (1998) to the capacitated vehicle routing problem and the vehicle routing problem with time windows. It was later applied to the pick-up and delivery vehicle routing problem with time windows (Bent and Van Hentenryck, 2006) with great results. Other methods that exhibit a similar framework to LNS include the ruin and recreate method (Schrimpf et al., 2000) and the iterated greedy heuristic (Ruiz and Stützle, 2007). An LNS extension is the adaptive LNS (ALNS) proposed by Ropke and Pisinger (2006a). The difference between LNS and ALNS is that it is possible to choose from among a number of different operators to destroy and repair a solution in ALNS, and the choice of a specific operator is determined based on the operator's performance.

The main ideas of our H-LNS are as follows. At each iteration, q stations are removed from the solution by a removal operator, and these q stations are put in a pool together with any previously unassigned stations, with q stations selected from the pool then added back to the solution using an insertion operator. There are a total of five removal operators and five insertion operators. These operators rely on simple mechanisms (e.g., greediness, randomness, biased sampling, noise, regret) to remove stations and reinsert them into the solution. One operator from each category is randomly chosen and applied to the solution. The new solution is improved by a simple tabu search if the solution is no more than $\lambda\%$ worse than the best known solution x^*. A solution is accepted as the new current solution only if its solution value is better than that of the current solution. The method's algorithmic framework is depicted in Algorithm 1. The differences between our method and ALNS are that (1) in our method the choice of a specific operator is not guided by the operator's performance, but rather by the uniform distribution, and (2) our method requires fewer parameters.

neighborhood search = heuristic by insertion and removal?

The input to Algorithm 1 is constructed by Algorithm 2. In Algorithm 1, there are removal and insertion operators (lines 3 and 4) and tabu search and intensification procedures (line 12). The removal and insertion operators are described in Sections 3.3 and 3.4, respectively, and tabu search and intensification procedures are depicted primarily by Algorithms 3-6. Note that the construction of an initial solution, neighborhood operators (including the removal and insertion operators), tabu search, and intensification procedures in our heuristic are highly reliant on knowledge of the station characteristics. For example, if a station is a pick-up station, in a neighbor solution generated by any one of the proposed operators, the drop-off quantity must be zero and the vehicle's pick-up quantity cannot be larger than the number of bikes available at the station or the available space in the vehicle. To obtain a station's pick-up or drop-off quantity, the neighborhood operators, construction heuristic, tabu search, and intensification

Algorithm 1 `H-LNS`

Require: A feasible initial solution x_0

Set $\bar{x} = x_0$ and $x^* = x_0$.
while $iterWI < K$ **do**
  $\hat{x} =$`removals`$(\bar{x})$
  $x =$`insertions`$(\hat{x})$
  if $z(x) < z(x^*)$ **then**
    Set $iterWI = 0$.
    Set $x^* = x$.
  else
    Set $iterWI = iterWI + 1$.
  end if
  if $z(x) < z(x^*) \times (1 + \lambda)$ **then**
    $x =$`tabuSearch`(x)
    if $z(x) < z(x^*)$ **then**
      Set $x^* = x$.
    end if
  end if
  if $z(x) < z(\bar{x})$ **then**
    Set $\bar{x} = x$.
  end if
end while

K=?

What is z function?

z function smaller = good?

if line 5 is true
x -> x*
then 11 is also true?

update standard

procedures must have knowledge of the station characteristics to select an appropriate rule to generate a feasible solution. All of these rules have been incorporated into Algorithms 2-6.

3.1 Solution representation

A solution x is made up of a set of vehicle routes and a set of loading and unloading quantities, where each route v is represented as $(i_0, i_1, \ldots, i_{n_v})$, $i_0 = i_{n_v} = 0$ and $i_1, \ldots, i_{n_v - 1} \in \mathcal{N}_0$. A node is denoted as i_h, where $h = 1, \ldots, n_v - 1$ is the placement of i_h in route v. For each node i_h, there is also a $y^+_{i_h v}$ or $y^-_{i_h v}$ associated with it (depending on whether $i_h \in \mathcal{P}$ or $i_h \in \mathcal{D}$). Solution x is evaluated by an evaluation function $z(x) = \sum_{i \in \mathcal{N}} f_i(s_i) + \alpha \sum_{i \in \mathcal{N}_0} \sum_{j \in \mathcal{N}_0, j \neq i} \sum_{v \in \mathcal{V}} t_{ij} x_{ijv}$, where s_i is defined by (2).

route solution

each node 2 decision variable

penalty function + parameter * travel time

3.2 Construction heuristic

The construction heuristic generates one route at a time. Pick-up stations are first assigned to a route, and the number of bikes to be picked up from each station is determined by a number of factors: 1) spare capacity, 2) the remaining time of the repositioning operation (τ), 3) the initial inventory level s_i^0, and 4) the optimal inventory level s_i^I. If no more pick-up stations can be assigned, then drop-off stations are inserted. The number of bikes to be dropped off at each station is determined by the vehicle load, s_i^0, and s_i^I. The heuristic alternates between the assignment of pick-up stations and drop-off stations until it is not possible to assign any more stations without violating a constraint. The steps of the algorithm can be found in Algorithm 2. The heuristic is adapted from the one in Ho and Szeto (2014) to take into account multiple vehicle routes and split pick-ups/drop-offs.

Algorithm 2 `Construction`

Require: Let PL (DL) be an ordered set of pick-up (drop-off) nodes and PL_d (DL_d) the dth node of PL (DL).

Difference of (Optimal - initial) largest -> 排前面 ? function f?

Sort PL such that $d > \bar{d} \Rightarrow |f_{PL_d}(s^0_{PL_d}) - f_{PL_d}(s^I_{PL_d})| \geq |f_{PL_{\bar{d}}}(s^0_{PL_{\bar{d}}}) - f_{PL_{\bar{d}}}(s^I_{PL_{\bar{d}}})|$.
Sort DL such that $d > \bar{d} \Rightarrow |f_{PL_d}(s^0_{PL_d}) - f_{PL_d}(s^I_{PL_d})| \geq |f_{PL_{\bar{d}}}(s^0_{PL_{\bar{d}}}) - f_{PL_{\bar{d}}}(s^I_{PL_{\bar{d}}})|$.
for $v = 1, \ldots, |\mathcal{V}|$ **do**
  Set $d = 1$ and $w = 1$. Initialize the route as $(i_0, \ldots, i_{n_v})$.
  Set $\tau = T$ and $y_{i_{n_v-1} i_{n_v} v} = 0$.

min(vehicle capacity - existing vehicle, initial - optimal level, ?the third one? Why need to use U: Time to unload a bike)

  repeat
    repeat
      Set $g = PL_d$

remaining time - used time

      Set $\tau_1 = \tau$ and $\tau = \tau - t_{i_{n_v-1} g} - t_{g i_{n_v}} + t_{i_{n_v-1} i_{n_v}}$

i0 and i nv = node 0

      Set $y^P_{gv} = \min\{k - y_{i_{n_v-1} i_{n_v} v}, s^0_g - s^I_g - \sum_r y^P_{gr}, \lfloor \tau/(U+L) \rfloor\}$
      if $y^P_{gv} > 0$ **then**

去尾以嘅node 變成去difference最大嘅node

        Node g is inserted between i_{n_v-1} and i_{n_v} in route v. Set $\tau = \tau - (U+L) y^P_{gv}$.
        Set $y_{i_{n_v-1} g v} = y_{i_{n_v-1} i_{n_v} v}$ and $y_{g i_{n_v} v} = y_{i_{n_v-1} g v} + y^P_{gv}$.
      else

帶bike數目 轉去完g先去final

        Set $\tau = \tau_1$.
      end if
      Set $d = d + 1$
    until no more nodes $g \in PL$ can be added without violating the constraints or $d > |PL|$
    repeat
      Set $g = DL_w$
      Set $\tau_1 = \tau$ and $\tau = \tau - t_{i_{n_v-1} g} - t_{g i_{n_v}} + t_{i_{n_v-1} i_{n_v}}$
      Set $y^D_{gv} = \min\{y_{i_{n_v-1} i_{n_v} v}, s^I_g - s^0_g - \sum_r y^D_{gr}\}$
      if $y^D_{gv} > 0$ and $\tau \geq 0$ **then**
        Node g is inserted between i_{n_v-1} and i_{n_v} in route v.
        Set $y_{i_{n_v-1} g v} = y_{i_{n_v-1} i_{n_v} v}$ and $y_{g i_{n_v} v} = y_{i_{n_v-1} g v} - y^D_{gv}$.
      else
        Set $\tau = \tau_1$.
      end if
      Set $w = w + 1$
    until no more nodes $g \in DL$ can be added without violating the constraints or $w > |DL|$
  until not possible to add more nodes without violating the constraints
end for

3.3 Removal operators

The five following removal operators are selected randomly in H-LNS.

Random removal 1 The following is repeated q times: randomly choose a vehicle route v; randomly choose a node from route v, and remove it from the route.

Random removal 2 The following is repeated q times: randomly choose a vehicle route v; remove node i from route v that contributes the least to the objective value, i.e., $i = \arg\min_{j \in W_v \cap P}\{f_j(s_j^0 - \sum_r y_{jr}^P + y_{jv}^P) - f_j(s_j^0 - \sum_r y_{jr}^P)\}$ or $i = \arg\min_{j \in W_v \cap D}\{f_j(s_j^0 + \sum_r y_{jr}^D - y_{jv}^D) - f_j(s_j^0 + \sum_r y_{jr}^D)\}$, where W_v denotes the set of nodes visited by route v. Note that the pick-up/drop-off quantities of the remaining nodes remain unchanged.

Cluster removal The idea is to partition the nodes of a randomly chosen route v into two clusters using an algorithm to find the minimum spanning tree (Kruskal, 1956). Instead of executing the algorithm until the end, the modified version terminates when there are two connected components left. Then, a cluster is selected at random, and its nodes are removed. If the number of nodes removed is less than q, then one of the removed nodes is chosen at random (say node i). The closest visited node j^* of node i belonging to route r ($r \neq v$) is determined, where $j^* = \arg\min_{j \in W_r \setminus W_v} t_{ij}$. The nodes of route r is partitioned into two clusters. The entire procedure is repeated until at least q nodes are removed. This method was first applied by Ropke and Pisinger (2006b) to vehicle routing problems with backhauls.

Radial ruin This operator's aim is to remove q nodes from the solution. It was first mentioned in Schrimpf et al. (2000). First, randomly select node i (among those being visited). Second, remove i and its $q-1$ nearest (in terms of the shortest travel time) neighbors from the solution.

Neighbor graph removal This removal operator makes use of historical information to destroy a solution (Ropke and Pisinger, 2006b). The historical information is saved in a directed graph, where nodes correspond to stations and the edge weights correspond to the objective function value of the best solution. Initially, all weights are set to infinity. Once a new best solution x^* is found, all of the arcs found in x^* have their weights updated with $z(x^*)$. Based on the current solution $\bar{x}$, this operator computes a score for each of the nodes in $\bar{x}$. The score of node i is the summed weights of the arcs incident to i. The nodes with large scores are most likely misplaced and are selected for removal. The removals are randomly determined, but controlled by the parameter $\phi \geq 1$. As the nodes are sorted in descending order according to their scores (stored in a list Ω), the higher ϕ is set, the greater the chance of the nodes with the larger scores being selected. The index p of the node is first determined based on $\lfloor \mathcal{U}(0,1)^{\phi} |\Omega| \rfloor$, and the node to be removed is denoted as Ω_p. The scores of the nodes incident to Ω_p are then updated before another node is chosen for removal. This process is repeated until q nodes are removed.

arc weight high -> touring, high score = high chance to mismatch, but not highest score then remove, but random depending on phi

When a node is removed from the solution, the solution may no longer remain feasible w.r.t. the loading and capacity constraints. Hence, the solution is repaired after q nodes are removed by assigning the pick-up/drop-off quantity at each node with the minimal number of bikes to render the solution feasible. Then, the pick-up/drop-off quantities are gradually increased until it is not possible to increase them any further without violating the loading and capacity constraints (see Appendix A for details).

Why need 5 removal? Random removal

3.4 Insertion operators

Any node in the pool Λ is subject to potential insertion in the solution. Λ consists of all unassigned nodes, nodes that were previously removed by one of the removal operators, and nodes whose $s_i^0 - \sum_r y_{ir}^P \neq s_i^I$ or $s_i^0 + \sum_r y_{ir}^D \neq s_i^I$. The latter implies that a node that has already been visited by route r may be allowed to be inserted into route v (where $v \neq r$). The five following operators are used randomly to insert nodes.

Time-based insertion with noise The following is repeated q times: Choose node $i^* = \arg\min_{i \in \Lambda}\{\delta_i + \zeta_i\}$, where δ_i is the additional travel time for inserting i between two consecutive nodes using the cheapest insertion criterion, and the evaluation function is perturbed with some noise ζ_i. The noise ζ_i is drawn randomly from the range $[-\rho\eta_{\max}, \rho\eta_{\max}]$, where $\eta_{\max} = \max_{i,j \in \mathcal{N}_0}\{t_{ij}\}$ and ρ is a parameter (Charon and Hudry, 1993).

Best insertion The following is repeated q times: Node $i^* = \arg\min_{i \in \Lambda} z(x \cup \{i\})$ is selected and inserted in the best position between two consecutive nodes in a route. For this insertion to be feasible, all constraints have to be respected and either $y_{i^*v}^P > 0$ or $y_{i^*v}^D > 0$ has to be true. To avoid recomputing all of the pick-up/drop-off quantities of the nodes in a route, it suffices to adjust the quantities at the depot, of the preceding or succeeding node of the inserted node i^*. This operator is also the insertion operator used in tabu search. Details on how to determine the values of $y_{i^*v}^P$ and $y_{i^*v}^D$ and how to adjust the other nodes' quantities can be found in Section 3.5 and Algorithm 4.

Biased random insertion A node is selected for insertion based on the geometric distribution. The candidates (for potential insertion) in the list Ω are sorted in ascending order of their insertion cost (i.e., the additional travel time required to visit the node). Each candidate is assigned a probability from the geometric distribution (i.e., $\beta(1-\beta)^p$, where $\beta \in [0.05, 0.25]$ and p is the index of the candidate in Ω). β can be interpreted as the probability of selecting the candidate with the lowest insertion cost at one specific iteration of this procedure (Juan et al., 2010). The selection process is repeated until q nodes are inserted.

Regret insertion A node i^* is selected for insertion based on its regret value, which is the cost difference between its best insertion position and its $\hat{k}$-th best insertion position. It is the best to insert nodes with high regret values first; otherwise it may be difficult to insert them at a later stage owing to the lack of favorable insertion positions. A node i^* is selected from among the set Λ based on $i^* = \arg\max_{i \in \Lambda} \sum_{e=1}^{\hat{k}} (\delta_i^e - \delta_i^1)$, where δ_i^e is the additional travel time induced after inserting i into the e-th best position in the solution. Note that this step is a bit different from that in Ropke and Pisinger (2006a), as they restricted the different insertion positions to being in different routes, whereas in our case the best insertion positions can be in the same route. This procedure is repeated q times. $\hat{k}$ is set to 3 in our implementation, as this value has been shown to produce good results in the literature on various routing problems.

Random insertion Randomly select a station, and insert it in its best position (with the least additional travel time induced) in the solution. This procedure is repeated q times.

After q nodes are inserted, the values of y_{iv}^P or y_{iv}^D (of these q nodes) are all set to zero except for the operator "Best insertion". Sometimes the objective value can be improved by reallocating the bikes between the nodes of a vehicle route. Hence, the pick-up and/or drop-off quantities of every route are adjusted to obtain a better objective value. Details on how this is achieved can be found in Section 3.6 and in Algorithm 6.

3.5 Tabu search

After q times removal & insertion?

Tabu search (TS) is applied to promising solutions (i.e., solutions that are no worse than $\lambda\%$ of the current best known solution x^*). TS selects the best non-tabu solution (or a solution that satisfies the aspiration criterion) from a neighborhood comprising the neighbor solutions obtained by three operators, namely, the removal (removes a station from its route), insertion (inserts a station) and replacement (replaces a pick-up (drop-off) station with another pick-up (drop-off) station) operators. The neighborhood consists of feasible solutions only. A neighbor solution is created when the routing sequence is altered by one of the three operators. To ensure that the neighbor solution is feasible, the number of bikes to be picked up/dropped off at the preceding and succeeding stations of the inserted/removed station must be updated simultaneously. The reverse move of the new incumbent solution is set tabu for a number of iterations. TS is run for a fixed number of iterations, γ.

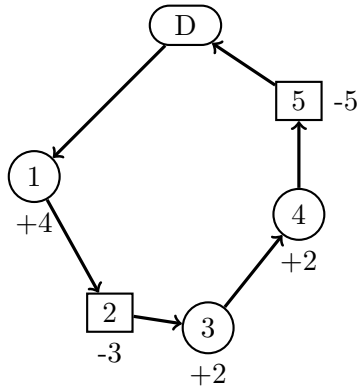
The afore-described TS is applied to the single-vehicle bike-repositioning problem in Ho and Szeto (2014). In this paper, we modify the algorithm to conform with the features of a multi-vehicle bike-repositioning problem in which multiple visits to a node by different vehicles are accounted for when creating the neighborhood. In the following, we highlight the changes that have been made and illustrate the concepts by giving examples (Figures 1-3).

Neighborhood $N_1(x)$ consists of all feasible neighbor solutions obtained by applying the removal operator, which removes one node i_h from route v at a time (such that the resulting route then continues to i_{h+1}, bypassing i_h from i_{h-1}), and then adjusts the pick-up or drop-off quantities at the depot ($y^P_{i_0v}$), the preceding station ($y^P_{i_{h-1}v}$ or $y^D_{i_{h-1}v}$), or the succeeding station ($y^P_{i_{h+1}v}$ or $y^D_{i_{h+1}v}$) to make the neighbor solution $\bar{x}$ feasible (if possible). (Note that the loading and unloading quantities at the other nodes remain unchanged.) The efficiency lies in being able to evaluate the neighborhood without recomputing the pick-up/drop-off quantity of every station in a neighbor solution, by simply adjusting the quantity of one of the three aforementioned stations. For each node that is removed from the solution, up to three feasible neighbor solutions are included in $N_1(x)$. The necessary conditions and steps of this adjustment are stated in Algorithm 3, where $\tilde{y}^P_{iv}$ ($\tilde{y}^D_{iv}$) represents y^P_{iv} (y^D_{iv}) in neighbor solution $\bar{x}$. The adjustment of $y^P_{i_0v}$, $y^P_{i_{h-1}v}$ or $y^D_{i_{h-1}v}$, or $y^P_{i_{h+1}v}$ or $y^D_{i_{h+1}v}$, depends on whether i_h is a pick-up node (Algorithm 3, lines 3-21) or a drop-off node (Algorithm 3, lines 22-41). Obviously, in addition to satisfying the conditions stated in the algorithm, a feasible neighbor solution also needs to obey the time constraint. The necessary changes are the additions of the summations of the respective pick-up/drop-off quantities to account for the possibility of a node being visited by more than one vehicle.

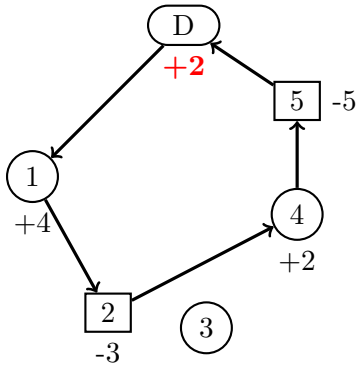
If i_h is a pick-up node, then the first potential neighbor solution consists of a solution whereby vehicle v picks up $y^P_{i_hv}$ extra bikes at the depot (lines 4-7 and Figure 1b) rather than from i_h (Figure 1a). For this solution to be feasible, the condition in line 4 has to be satisfied. $\tilde{y}^P_{i_0v}$ in line 6 represents the adjusted number of bikes to be picked up from the depot if this solution is chosen and implemented. Similar notations (those with $\sim$) are used later in the paper and in Algorithms 4 and 5 to denote the adjusted quantities. The second potential neighbor solution is obtained by adjusting the preceding station's pick-up/drop-off quantity. If i_{h-1} is also a pick-up node, then i_{h-1} needs to provide $y^P_{i_hv}$ extra bikes for pick-up (lines 8-11). If i_{h-1} is a drop-off node, then the vehicle needs to drop off $y^P_{i_hv}$ fewer bikes at i_{h-1} (lines 12-16 and Figure 1c). The third potential neighbor solution is reached by adjusting the succeeding station's pick-up/drop-off quantity. The logic behind lines 17-21 (see Figure 1d) follows the same principles as lines 8-16. Lines 22-41 describe the possibilities of creating neighbor solutions when i_h is a drop-off node, and they follow similar patterns to lines 3-21.

Algorithm 3 The removal move

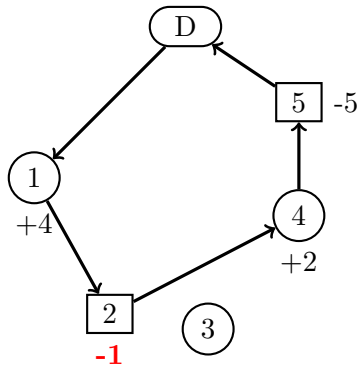
1: **for** $v = 1, \ldots, |\mathcal{V}|$ **do**
2: **for** $h = 1, \ldots, n_v - 1$ **do**
3: **if** $i_h \in W_v \cap \mathcal{P}$ **then**
4: $\vartheta = \min_{l=0,1,\ldots,h-1}\{k - y_{i_l i_{l+1} v}\}$
5: **if** ($\vartheta \geq y^P_{i_h v}$ **and** $y^P_{i_0 v} + y^P_{i_h v} \leq k$ **and** $s^0_{i_0} - \sum_r y^P_{i_0 r} - y^P_{i_h v} \geq 0$) **then**
6: Set $\tilde{y}^P_{i_0 v} = y^P_{i_0 v} + y^P_{i_h v}$ and $\tilde{y}^P_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
7: **end if**
8: **if** $i_{h-1} \in W_v \cap \mathcal{P}$ **then**
9: **if** ($s^0_{i_{h-1}} - \sum_r y^P_{i_{h-1} r} - y^P_{i_h v} \geq 0$) **then**
10: Set $\tilde{y}^P_{i_{h-1} v} = y^P_{i_{h-1} v} + y^P_{i_h v}$ and $\tilde{y}^P_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
11: **end if**
12: **else if** $i_{h-1} \in W_v \cap \mathcal{D}$ **then**
13: **if** ($s^0_{i_{h-1}} + \sum_r y^D_{i_{h-1} r} - y^P_{i_h v} \leq c_{i_{h-1}}$ **and** $y^D_{i_{h-1} v} > y^P_{i_h v}$) **then**
14: Set $\tilde{y}^D_{i_{h-1} v} = y^D_{i_{h-1} v} - y^P_{i_h v}$ and $\tilde{y}^P_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
15: **end if**
16: **end if**
17: **if** $i_{h+1} \in W_v \cap \mathcal{P}$ **then**
18: Same as lines 9-11, except replace $h - 1$ with $h + 1$.
19: **else if** $i_{h+1} \in W_v \cap \mathcal{D}$ **then**
20: Same as lines 13-15, except replace $h - 1$ with $h + 1$.
21: **end if**
22: **else if** $i_h \in W_v \cap \mathcal{D}$ **then**
23: $u = \min_{l=0,1,\ldots,h-1}\{y_{i_l i_{l+1} v}\}$
24: **if** ($u - y^D_{i_h v} \geq 0$ **and** $y^P_{i_0 v} - y^D_{i_h v} \geq 0$) **then**
25: Set $\tilde{y}^P_{i_0 v} = y^P_{i_0 v} - y^D_{i_h v}$ and $\tilde{y}^D_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
26: **end if**
27: **if** $i_{h-1} \in W_v \cap \mathcal{D}$ **then**
28: **if** ($s^0_{i_{h-1}} + \sum_r y^D_{i_{h-1} r} + y^D_{i_h v} \leq c_{i_{h-1}}$) **then**
29: Set $\tilde{y}^D_{i_{h-1} v} = y^D_{i_{h-1} v} + y^D_{i_h v}$ and $\tilde{y}^D_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
30: **end if**
31: **else if** $i_{h-1} \in W_v \cap \mathcal{P}$ **then**
32: **if** ($s^0_{i_{h-1}} - \sum_r y^P_{i_{h-1} r} + y^D_{i_h v} \geq 0$ **and** $y^P_{i_{h-1} v} > y^D_{i_h v}$) **then**
33: Set $\tilde{y}^P_{i_{h-1} v} = y^P_{i_{h-1} v} - y^D_{i_h v}$ and $\tilde{y}^D_{i_h v} = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_1(x)$.
34: **end if**
35: **end if**
36: **if** $i_{h+1} \in W_v \cap \mathcal{D}$ **then**
37: Same as lines 28-30, except replace $h - 1$ with $h + 1$.
38: **else if** $i_{h+1} \in W_v \cap \mathcal{P}$ **then**
39: Same as lines 32-34, except replace $h - 1$ with $h + 1$.
40: **end if**
41: **end if**
42: **end for**
43: **end for**



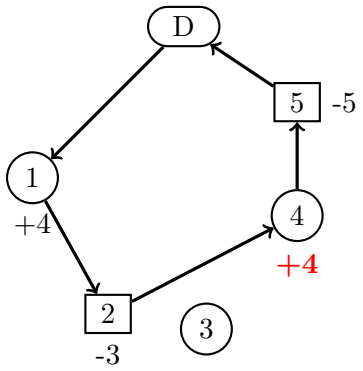
(a) The original route with five stations (with circles denoting the pick-up stations, and squares the drop-off stations). The numbers below the nodes are the pick-up (+) and drop-off (-) quantities.



(b) The new route with station 3 removed and an adjusted pick-up quantity for the depot.



(c) The new route with station 3 removed and an adjusted drop-off quantity for station 2.



(d) The new route with station 3 removed and an adjusted pick-up quantity for station 4.

Figure 1: Illustration of three neighbor solutions obtained by the removal operator when $h = 3$ and $i_h = 3$.

Can the neighbor sol. consists of multiple adjustment?

Neighborhood $N_2(x)$ consists of all feasible neighbor solutions obtained by applying the insertion operator. This operator inserts node $i \in \Lambda$ in every potential position (between every pair of adjacent nodes i_{h-1} and i_h) of every route, and then adjusts the pick-up/drop-off quantities at the depot ($y^P_{i_0 v}$), the preceding node ($y^P_{i_{h-1} v}$ or $y^D_{i_{h-1} v}$), or the succeeding node ($y^P_{i_h v}$ or $y^D_{i_h v}$) to make the resulting neighbor solution feasible. Note that the quantities of the other nodes remain unchanged. To account for the possibility of multiple trips in a solution, node i can also be inserted after depot i_{n_v}, but then an additional depot must be added after i to make it feasible. For each insertion of i between i_{h-1} and i_h, up to four neighbor solutions are included in $N_2(x)$: 1) no adjustment of existing pick-up/drop-off quantities, 2) $y^P_{i_0 v}$ is adjusted, 3) $y^P_{i_{h-1} v}$ or $y^D_{i_{h-1} v}$ is adjusted, and 4) $y^P_{i_h v}$ or $y^D_{i_h v}$ is adjusted. A feasible neighbor solution needs to satisfy all of the constraints. The necessary conditions regarding the time, capacities, and load of route v are listed in Algorithm 4. The adjustment of $y^P_{i_0 v}$, $y^P_{i_{h-1} v}$ or $y^D_{i_{h-1} v}$, or $y^P_{i_h v}$ or $y^D_{i_h v}$ depends on whether i_h is a pick-up node (Algorithm 4, lines 5-35 and 69-74) or a drop-off node (Algorithm 4, lines 36-67 and 75-81). The applied changes are the summations of the pick-up/drop-off quantities to account for multiple visits to a node. Lines 69-74 are new to Algorithm 4. They are where a new trip is created by adding a pick-up node in position $n_v + 1$ after the depot located in position n_v of route v. To render the insertion feasible, an extra depot-node needs to be inserted in position $n_v + 2$.

If i is a drop-off node, then the first potential neighbor solution is to insert i between i_{h-1} and i_h, and then determine the number of bikes to drop off at i while satisfying all of the constraints (lines 37-41). A second potential neighbor solution is to let the vehicle pick up y^D_{iv} extra bikes at the depot so that it can drop them off at i (lines 42-47 and Figure 2b). The third potential neighbor solution is to adjust the preceding station's pick-up/drop-off quantity. If i_{h-1} is also a drop-off node, then the vehicle can drop off y^D_{iv} fewer bikes at i_{h-1} and then drop off the remainder at i (lines 48-52). If i_{h-1} is a pick-up node, then, for every additional bike that is picked up from i_{h-1}, a bike is dropped off at i (lines 53-58 and Figure 2c). The fourth potential solution is to adjust the succeeding station's pick-up/drop-off quantity. The logic behind lines 59-60 follows the same principle as lines 49-52. If i_h is a pick-up node, then, for every bike that is dropped off at i, an extra bike is picked up from i_h (lines 61-66 and Figure 2d). Because it is possible for a vehicle to return to the depot more than once during the repositioning duration, a possible neighbor solution with an extra trip is shown in Figure 2e (lines 75-81), where the vehicle picks up $y^P_{i_{n_v} v}$ bikes from the depot and drops them off at i.

Neighborhood $N_3(x)$ consists of all feasible neighbor solutions obtained by applying the exchange operator to solution x. This operator replaces node i_h in route v of solution x with another node $i \in \Lambda$. In addition to respecting the time constraint, i_h and i have to be the same type and $i \notin W_v$. Ho and Szeto (2014) restricted $i_h \in W_v$ but, because vehicle v can perform more than one trip, we allow $i_h \in W_v \cup \{0\}$. If i_h is the depot, then it may be exchanged by either a pick-up or drop-off node. The pick-up/drop-off quantity of i is the same as that of i_h, given the neighbor solution remains feasible. The necessary conditions are in Algorithm 5. The applied changes are the inclusion of the summations of the pick-up/drop-off quantities to account for a node being visited more than once. Figure 3 shows the potential neighbor solutions when one pick-up station is replaced by another ($i = 6$).

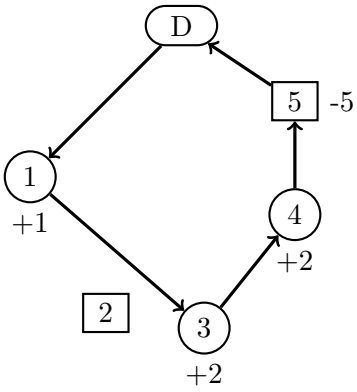
add/remove the node after each move

After a move is implemented, the pool Λ may be updated. If the chosen move is the removal move, then node i_h is added to Λ if it has not already been included. If the chosen move is the insertion move, then node i is removed from Λ if $s^0_i - \sum_r y^P_{ir} = s^I_i$ (where $i \in \mathcal{P}$) or $s^0_i + \sum_r y^D_{ir} = s^I_i$ (where $i \in \mathcal{D}$). Finally, if the chosen move is the exchange move, then the foregoing rules apply to both i_h and i.

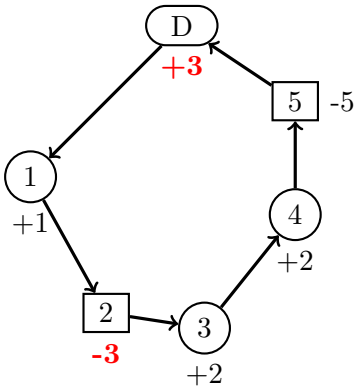
Algorithm 4 The insertion move

1: **for** $i \in \Lambda$ **do**
2: **for** $v = 1, \ldots, |\mathcal{V}|$ **do**
3: $\tau_1 = T-$ LHS of (10)
4: **for** $h = 1, \ldots, n_v$ **do**
5: **if** $i \in \mathcal{P} \setminus W_v$ **then**
6: $\tau = \tau_1 - (t_{i_{h-1}i} + t_{ii_h} - t_{i_{h-1}i_h})$
7: $\kappa = \max_{l=h-1,h,\ldots,n_v-1}\{y_{i_l i_{l+1} v}\}$
8: $y = \min\{\lfloor \tau/(U+L) \rfloor, s_i^0 - \sum_r y_{ir}^P, k - \kappa\}$
9: **if** $y > 0$ **then**
10: Set $\tilde{y}_{iv}^P = y$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
11: **end if**
12: $u = \min_{l=0,\ldots,h-1}\{y_{i_l i_{l+1} v}\}$
13: $y = \min\{s_i^0 - \sum_r y_{ir}^P, u\}$
14: **if** $y > 0$ **then**
15: Set $\tilde{y}_{iv}^P = y$ and $\tilde{y}_{i_0 v}^P = y_{i_0 v}^P - \tilde{y}_{iv}^P$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
16: **end if**
17: **if** $i_{h-1} \in W_v \cap \mathcal{P}$ **then**
18: $y = \arg\min_{\omega=0,1,\ldots,\min\{s_i^0, y_{i_{h-1}v}^P\}} f_{i_{h-1}}(s_{i_{h-1}}^0 - \sum_r y_{i_{h-1}r}^P + \omega) + f_i(s_i^0 - \sum_r y_{ir}^P - \omega)$
19: **if** $y > 0$ **and** $y_{i_{h-1}v}^P > y$ **then**
20: Set $\tilde{y}_{iv}^P = y$ and $\tilde{y}_{i_{h-1}v}^P = y_{i_{h-1}v}^P - \tilde{y}_{iv}^P$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
21: **end if**
22: **else if** $i_{h-1} \in W_v \cap \mathcal{D}$ **then**
23: $y = \min\{y_{i_{h-1}i_h v}, c_{i_{h-1}} - s_{i_{h-1}}^0 - \sum_r y_{i_{h-1}r}^D, s_i^0 - \sum_r y_{ir}^P, \lfloor \tau/(U+L) \rfloor\}$
24: **if** $y > 0$ **then**
25: Set $\tilde{y}_{iv}^P = y$ and $\tilde{y}_{i_{h-1}v}^D = y_{i_{h-1}v}^D + \tilde{y}_{iv}^P$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
26: **end if**
27: **end if**
28: **if** $i_h \in W_v \cap \mathcal{P}$ **then**
29: Same as lines 18-21, except replace $h-1$ with h.
30: **else if** $i_h \in W_v \cap \mathcal{D}$ **then**
31: $y = \min\{k - y_{i_{h-1}i_h v}, c_{i_h} - s_{i_h}^0 - \sum_r y_{i_h r}^D, s_i^0 - \sum_r y_{ir}^P, \lfloor \tau/(U+L) \rfloor\}$
32: **if** $y > 0$ **then**
33: Set $\tilde{y}_{iv}^P = y$ and $\tilde{y}_{i_h v}^D = y_{i_h v}^D + \tilde{y}_{iv}^P$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
34: **end if**
35: **end if**

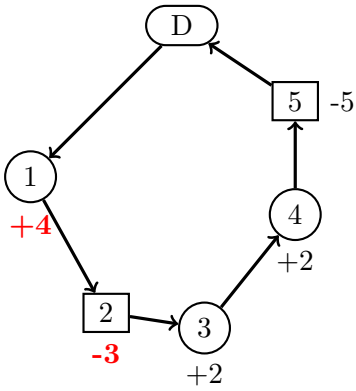
36: **else if** $i \in \mathcal{D} \setminus W_v$ **then**
37: $\mu = \min_{l=h-1,h,\ldots,n_v-1}\{y_{i_l i_{l+1} v}\}$
38: $y = \min\{c_i - s_i^0 - \sum_r y_{ir}^D, \mu\}$
39: **if** $y > 0$ **then**
40: Set $\tilde{y}_{iv}^D = y$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
41: **end if**
42: $\tau = \tau_1 - (t_{i_{h-1}i} + t_{ii_h} - t_{i_{h-1}i_h})$
43: $\vartheta = \min_{l=0,1,\ldots,h-1}\{k - y_{i_l i_{l+1} v}\}$
44: $y = \min\{\lfloor \tau/(U+L) \rfloor, c_i - s_i^0 - \sum_r y_{ir}^D, \vartheta, s_{i_0}^0 - \sum_r y_{i_0 r}^P\}$
45: **if** $y > 0$ **then**
46: Set $\tilde{y}_{iv}^D = y$ and $\tilde{y}_{i_0 v}^P = y_{i_0 v}^P + \tilde{y}_{iv}^D$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
47: **end if**
48: **if** $i_{h-1} \in W_v \cap \mathcal{D}$ **then**
49: $y = \arg\min_{\omega=0,1,\ldots,\min\{c_i - s_i^0 - \sum_r y_{ir}^D, y_{i_{h-1}v}^D\}} f_{i_{h-1}}(s_{i_{h-1}}^0 + \sum_r y_{i_{h-1}r}^D - \omega) + f_i(s_i^0 + \sum_r y_{ir}^D + \omega)$
50: **if** ($y > 0$ **and** $y_{i_{h-1}v}^D > y$) **then**
51: Set $\tilde{y}_{iv}^D = y$ and $\tilde{y}_{i_{h-1}v}^D = y_{i_{h-1}v}^D - \tilde{y}_{iv}^D$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
52: **end if**
53: **else if** $i_{h-1} \in W_v \cap \mathcal{P}$ **then**
54: $y = \min\{k - y_{i_{h-1}i_h v}, s_{i_{h-1}}^0 - \sum_r y_{i_{h-1}r}^P, c_i - s_i^0 - \sum_r y_{ir}^D, \lfloor \tau/(U+L) \rfloor\}$
55: **if** $y > 0$ **then**
56: Set $\tilde{y}_{iv}^D = y$ and $\tilde{y}_{i_{h-1}v}^P = y_{i_{h-1}v}^P + \tilde{y}_{iv}^D$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
57: **end if**
58: **end if**
59: **if** $i_h \in W_v \cap \mathcal{D}$ **then**
60: Same as lines 49-52, except replace $h-1$ with h.
61: **else if** $i_h \in W_v \cap \mathcal{P}$ **then**
62: $y = \min\{y_{i_{h-1}i_h v}, c_i - s_i^0 - \sum_r y_{ir}^D, s_{i_h}^0 - \sum_r y_{i_h r}^P, \lfloor \tau/(U+L) \rfloor\}$
63: **if** $y > 0$ **then**
64: Set $\tilde{y}_{iv}^D = y$ and $\tilde{y}_{i_h v}^P = y_{i_h v}^P + \tilde{y}_{iv}^D$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
65: **end if**
66: **end if**
67: **end if**
68: **end for**
69: **if** $i \in \mathcal{P} \setminus W_v$ **then**
70: $\tau = \tau_1 - (t_{i_{n_v} i} + t_{i i_{n_v+2}})$
71: $y = \min\{k, s_i^0 - \sum_r y_{ir}^P, \lfloor \tau/(U+L) \rfloor\}$
72: **if** $y > 0$ **then**
73: Set $\tilde{y}_{iv}^P = y$ and $\tilde{y}_{i_{n_v+2}v}^D = y$, where $i = i_{n_v+1}$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
74: **end if**
75: **else if** $i \in \mathcal{D} \setminus W_v$ **then**
76: $\tau = \tau_1 - (t_{i_{n_v} i} + t_{i i_{n_v+2}})$
77: $y = \min\{c_i - s_i^0 - \sum_r y_{ir}^D, k, s_{i_0}^0 - \sum_r y_{i_0 r}^P, \lfloor \tau/(U+L) \rfloor\}$
78: **if** $y > 0$ **then**
79: Set $\tilde{y}_{i_{n_v} v}^P = y$, $\tilde{y}_{iv}^D = y$ and $\tilde{y}_{i_{n_v+2}v}^D = 0$, where $i = i_{n_v+1}$. Create $\bar{x}$, and add $\bar{x}$ to $N_2(x)$.
80: **end if**
81: **end if**
82: **end for**
83: **end for**



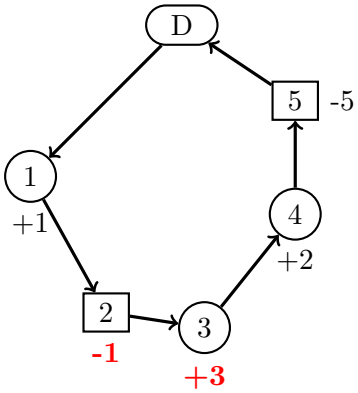
(a) The original route with four stations.



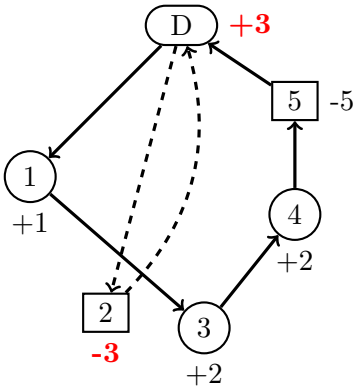
(b) The new route with station 2 inserted and an adjusted pick-up quantity for the depot.



(c) The new route with station 2 inserted and an adjusted pick-up quantity for station 1.



(d) The new route with station 2 inserted and an adjusted pick-up quantity for station 3.



(e) An additional trip is added, with the new trip being D-2-D and an adjusted pick-up quantity for the depot.

Figure 2: Illustration of four neighbor solutions obtained by the insertion operator when $i = 2$, $h = 2$, and $i_h = 3$.

Algorithm 5 The exchange move

for $i \in \Lambda$ **do**
  for $v = 1, \ldots, |\mathcal{V}|$ **do**
    $\tau_1 = T-$ LHS of (10)
    for $h = 1, \ldots, n_v - 1$ **do**
      $\tau = \tau_1 - t_{i_{h-1}i_h} - t_{i_h i_{h+1}} + t_{i_{h-1}i} + t_{ii_{h+1}}$
      if $i_h \in (W_v \cap \mathcal{P}) \cup \{0\}$ **and** $i \in \mathcal{P} \setminus W_v$ **and** $\tau \geq 0$ **then**
        if $s_i^0 - \sum_r y_{ir}^P - y_{i_h v}^P \geq 0$ **then**
          Set $\tilde{y}_{iv}^P = y_{i_h v}^P$ and $\tilde{y}_{i_h v}^P = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_3(x)$.
        end if
      else if $i_h \in (W_v \cap \mathcal{D}) \cup \{0\}$ **and** $i \in \mathcal{D} \setminus W_v$ **and** $\tau \geq 0$ **then**
        if $s_i^0 + \sum_r y_{ir}^D + y_{i_h v}^D \leq c_i$ **then**
          Set $\tilde{y}_{iv}^D = y_{i_h v}^D$ and $\tilde{y}_{i_h v}^D = 0$. Create $\bar{x}$, and add $\bar{x}$ to $N_3(x)$.
        end if
      end if
    end for
  end for
end for

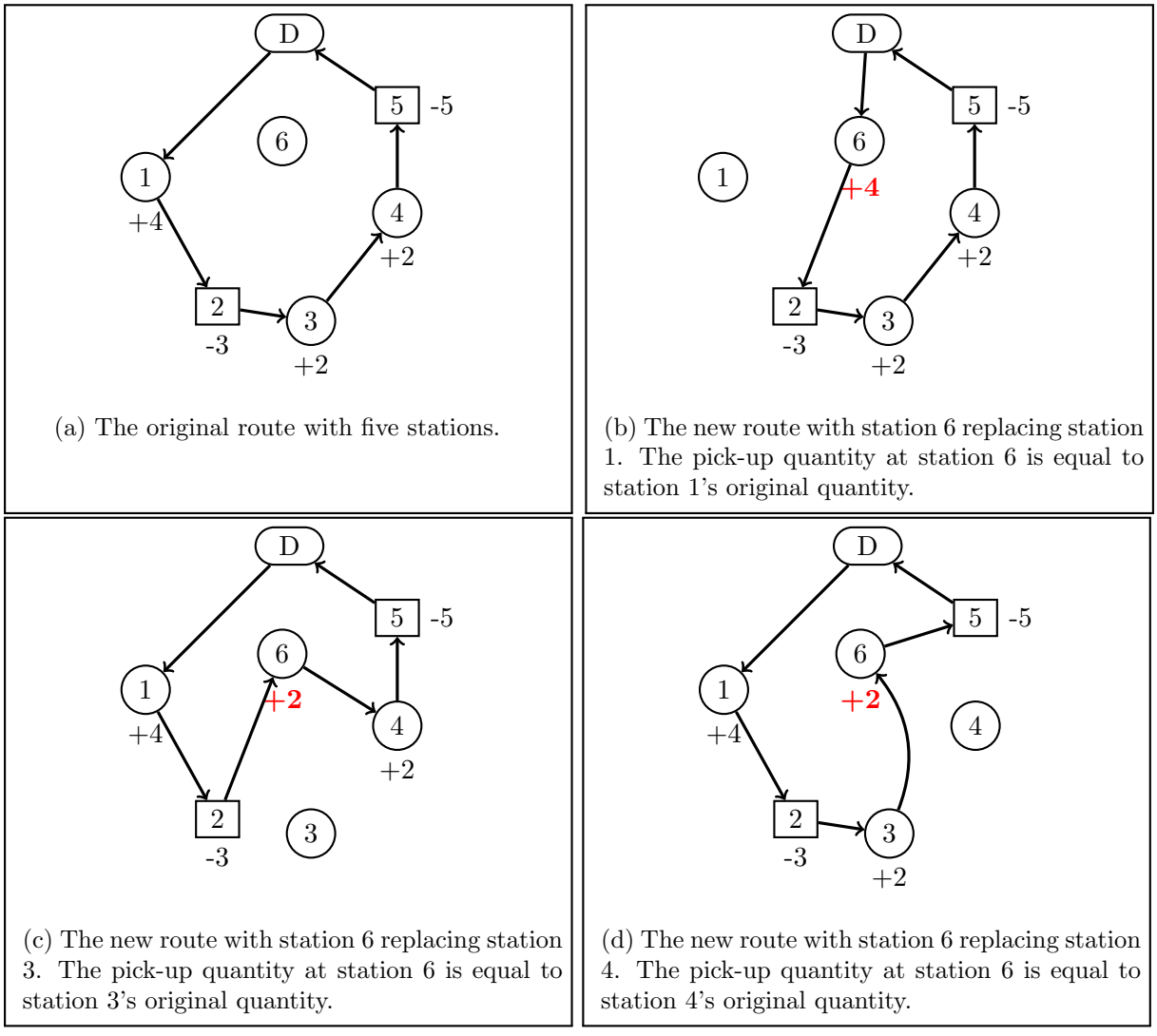


Figure 3: Illustration of the neighbor solutions produced by the exchange operator when $i = 6$.

3.6 Intensification procedures

The best solution found from TS is subject to further intensification procedures. Sometimes a solution can be improved by redistributing the assigned pick-up/drop-off quantities among the nodes of a given route. Note that this procedure does not increase the total time spent on the route, but simply alters the y^D_{iv}, y^P_{iv}, and y_{ijv} variables. The following approach was originally applied by Ho and Szeto (2014) to their single-vehicle repositioning problem. We have modified it to conform to the features of our multi-vehicle repositioning problem with split pick-ups/drop-offs. The resulting algorithm is given as Algorithm 6. Then, 2-opt (Lin, 1965) is applied to every vehicle route. This operator repeatedly removes two edges from the route, and adds back two new edges so that the route remains a tour. The objective is to reduce the total travel time, disregarding pick-up/drop-off quantities. Because the route is now (most likely) shorter, it is possible to add more nodes to the route. Using the insertion operator from Section 3.5, a maximum of five nodes are inserted. Finally, Algorithm 6 is applied to adjust the pick-up/drop-off quantities between the route's nodes.

The adjustment in Algorithm 6 is performed in a heuristic manner. To make it easier to comprehend, we focus here on four main cases of adjustment. In the first two cases, we consider one pair of nodes, i_m and i_h (where $m < h$), of the same type on route v (that is, either $i_m \in \mathcal{P} \cap W_v$ and $i_h \in \mathcal{P} \cap W_v$ or $i_m \in \mathcal{D} \cap W_v$ and $i_h \in \mathcal{D} \cap W_v$) at a time. The idea is to verify whether it is possible and beneficial to shift one or more (that is, y in the algorithm) units of the original quantity from i_m to i_h, or vice versa. Lines 4-12 describe the situation in which both i_m and i_h are pick-up nodes, whereas lines 13-22 describe that in which i_m and i_h are drop-off nodes. The third case concerns a pair of adjacent nodes (i_m, i_{m+1}), where i_m is the depot and $i_{m+1} \in \mathcal{P} \cap W_v$. As the depot can also be viewed as a pick-up node, we evaluate whether it is possible to pick up more (or less) from i_{m+1} and less (or more) from the depot. Lines 24-33 depict this situation. The last case concerns a pair of adjacent nodes (i_m, i_{m+1}), where $i_m \in \mathcal{D} \cap W_v$ and i_{m+1} is the depot. The idea is to evaluate the possibility of dropping off more bikes at i_m instead of at the depot. Lines 34-39 depict this situation.

4 Computational experiments

The heuristic was coded in C++, and all computational experiments were carried out on a Dell notebook with an Intel Core i5-2520M CPU@2.5 GHz. Three sets of instances were used to validate the efficiency and efficacy of the proposed H-LNS heuristic. The three datasets are as follows.

Set 1 This set contains instances of 75-200 stations and 2-3 vehicles with a vehicle capacity of 20. Two planning horizons were used: $T = 9000$ and $T = 18000$ seconds. The time for picking up or dropping off a bike was set to 60 seconds. The set contains 24 instances, and is available from `https://sites.google.com/site/drsinho/instances/hlns-set1.zip`.

Set 2 This set was first used by Forma et al. (2015), and its instances are available from `http://www.eng.tau.ac.il/~talraviv/Publications/3step%20data.zip`. The set contains instances of 75-200 stations and 2-3 vehicles with a vehicle capacity of 25. The time for picking up or dropping off a bike is 60 seconds, and planning horizon T is 18000 seconds. The data also differentiate between light, real, and heavy workloads (i.e., s^0_i is set closer to s^I_i when the workload is light rather than heavy). The set contains 30 instances.

Set 3 This set is based on the largest bike-sharing system in the United States, *Citi Bike* in New York. It contains instances based on geographical areas: Manhattan (302 stations);

Algorithm 6 Adjustment

for $v = 1, \ldots, |\mathcal{V}|$ **do**
  for $m = 0, \ldots, n_v - 1$ **do**
    for $h = m + 1, \ldots, n_v$ **do**
      if $i_m \in W_v \cap \mathcal{P}$ **and** $i_h \in W_v \cap \mathcal{P}$ **then**
        $y = \arg\min_{\omega=0,1,\ldots,\min\{y^P_{i_m v}, s^0_{i_h} - \sum_r y^P_{i_h r}\}} f_{i_m}(s^0_{i_m} - \sum_r y^P_{i_m r} + \omega) + f_{i_h}(s^0_{i_h} - \sum_r y^P_{i_h r} - \omega)$ subject to $y_{i_l i_{l+1} v} - \omega \geq 0, l = m, \ldots, h-1$
        if $y > 0$ **then**
          Set $y^P_{i_m v} = y^P_{i_m v} - y$, $y^P_{i_h v} = y^P_{i_h v} + y$ and $y_{i_l i_{l+1} v} = y_{i_l i_{l+1} v} - y$ $(l = m, \ldots, h-1)$.
        end if
        $y = \arg\min_{\omega=0,1,\ldots,\min\{y^P_{i_h v}, s^0_{i_m} - \sum_r y^P_{i_m r}, k - y_{i_m i_{m+1} v}\}} f_{i_m}(s^0_{i_m} - \sum_r y^P_{i_m r} - \omega) + f_{i_h}(s^0_{i_h} - \sum_r y^P_{i_h r} + \omega)$ subject to $y_{i_l i_{l+1} v} + \omega \leq k, l = m, \ldots, h-1$
        if $y > 0$ **then**
          Set $y^P_{i_m v} = y^P_{i_m v} + y$, $y^P_{i_h v} = y^P_{i_h v} - y$ and $y_{i_l i_{l+1} v} = y_{i_l i_{l+1} v} + y$ $(l = m, \ldots, h-1)$.
        end if
      else if $i_m \in W_v \cap \mathcal{D}$ **and** $i_h \in W_v \cap \mathcal{D}$ **then**
        $y = \arg\min_{\omega=0,1,\ldots,\min\{y^D_{i_m v}, c_{i_h} - s^0_{i_h} - \sum_r y^D_{i_h r}\}} f_{i_m}(s^0_{i_m} + \sum_r y^D_{i_m r} - \omega) + f_{i_h}(s^0_{i_h} + \sum_r y^D_{i_h r} + \omega)$ subject to $y_{i_l i_{l+1} v} + \omega \leq k, l = m, \ldots, h-1$
        if $y > 0$ **then**
          Set $y^D_{i_m v} = y^D_{i_m v} - y$, $y^D_{i_h v} = y^D_{i_h v} + y$ and $y_{i_l i_{l+1} v} = y_{i_l i_{l+1} v} + y$ $(l = m, \ldots, h-1)$.
        end if
        $y = \arg\min_{\omega=0,1,\ldots,\min\{y_{i_m i_{m+1} v}, y^D_{i_h v}, c_{i_m} - s^0_{i_m} - \sum_r y^D_{i_m r}\}} f_{i_m}(s^0_{i_m} + \sum_r y^D_{i_m r} + \omega) + f_{i_h}(s^0_{i_h} + \sum_r y^D_{i_h r} - \omega)$ subject to $y_{i_l i_{l+1} v} - \omega \geq 0, l = m, \ldots, h-1$
        if $y > 0$ **then**
          Set $y^D_{i_m v} = y^D_{i_m v} + y$, $y^D_{i_h v} = y^D_{i_h v} - y$ and $y_{i_l i_{l+1} v} = y_{i_l i_{l+1} v} - y$ $(l = m, \ldots, h-1)$.
        end if
      end if
    end for
    if $i_m = 0$ **and** $i_{m+1} \in W_v \cap \mathcal{P}$ **and** $y_{i_m i_{m+1} v} > 0$ **then**
      $y = \arg\min_{\omega=0,1,\ldots,\min\{s^0_{i_{m+1}} - \sum_r y^P_{i_{m+1} r}, y_{i_m i_{m+1} v}\}} f_{i_{m+1}}(s^0_{i_{m+1}} - \sum_r y^P_{i_{m+1} r} - \omega)$
      if $y > 0$ **then**
        Set $y^P_{i_{m+1} v} = y^P_{i_{m+1} v} + y$, $y^P_{i_m v} = y^P_{i_m v} - y$ and $y_{i_m i_{m+1} v} = y^P_{i_m v}$.
      else
        $y = \arg\min_{\omega=0,1,\ldots,\min\{k - y_{i_m i_{m+1} v}, y^P_{i_{m+1} v}\}} f_{i_{m+1}}(s^0_{i_{m+1}} - \sum_r y^P_{i_{m+1} r} + \omega)$
        if $y > 0$ **then**
          Set $y^P_{i_{m+1} v} = y^P_{i_{m+1} v} - y$, $y^P_{i_m v} = y^P_{i_m v} + y$ and $y_{i_m i_{m+1} v} = y^P_{i_m v}$.
        end if
      end if
    else if $i_m \in W_v \cap \mathcal{D}$ **and** $i_{m+1} = 0$ **and** $y_{i_m i_{m+1} v} > 0$ **then**
      $y = \arg\min_{\omega=0,1,\ldots,\min\{c_{i_m} - s^0_{i_m} - \sum_r y^D_{i_m r}, y_{i_m i_{m+1} v}\}} f_{i_m}(s^0_{i_m} + \sum_r y^D_{i_m r} + \omega)$
      if $y > 0$ **then**
        Set $y^D_{i_m v} = y^D_{i_m v} + y$, $y^D_{i_{m+1} v} = y^D_{i_{m+1} v} - y$ and $y_{i_m i_{m+1} v} = y^D_{i_{m+1} v}$.
      end if
    end if
  end for
end for

Manhattan and Jersey City (349 stations); Manhattan, Brooklyn, and Queens (471 stations); and all four areas (518 stations). Asymmetric travel times (in seconds) between the stations were obtained from the *Open Source Routing Machine*[1]. The convex penalty functions were derived from the methodology proposed by Raviv and Kolka (2013) using real-time data collected from the Citi Bike website[2]. The number of vehicles varies between 3 and 5, and the vehicle capacity is 25. The time for picking up or dropping off a bike is 60 seconds, and planning horizon T is 18000 seconds. The set contains 12 instances, and is available from `https://sites.google.com/site/drsinho/instances/hlns-set3.zip`.

Unless otherwise specified, α was set to zero in most test instances to ignore the influence of the second term, that is, total travel time (i.e., to ensure that minimizing the total penalty was of the greatest importance). α was set to 1/900 in some of the test instances in Section 4.4 because Forma et al. (2015) used that value, and we wanted to compare our heuristic with their math heuristic.

4.1 Parameter tuning

Initially, the parameters were set as follows: $\lambda = 0.1$, $\rho = 0.1$, $\phi = 3$, $K = 100$, $\gamma = 50$, and $q \in \{1, \lceil 0.1 \times |\mathcal{N}| \rceil\}$ (see Table 2). The three parameters to be tuned were λ, ρ, and ϕ. K and γ did not required tuning, as it is obvious that the larger their values are, the better the solutions that will be achieved, albeit at the expense of much greater computing time. Nevertheless, setting K and γ to the above values provides a good trade-off between solution quality and computing time. The larger q is, the more time needed to destroy and recreate a feasible solution. Hence, we decided to allow q to be randomly drawn from a range bounded by 10% of the instance size. The first parameter to tune was λ, which restricts the number of solutions that can be improved by the tabu search. λ was set to take values from $\{0.05, 0.1, 0.15, 0.2\}$. The larger the value λ is assigned, the more time is needed to execute the algorithm. By running each of the 24 instances in Set 1 20 times for each value from the range, it is found that setting $\lambda = 0.1$ yields the best average results. With λ set to 0.1, the next parameter to tune was ρ, which controls the level of noise to be added to an evaluation function in the insertion operator "Time-based insertion with noise". ρ was set to take values from $\{0.025, 0.05, 0.1, 0.2, 0.4\}$. The larger the value that ρ takes, the wider the range that noise is drawn from. Experiments showed that setting ρ to 0.1 yields the best average results. With $\lambda = 0.1$ and $\rho = 0.1$, the last parameter to tune was ϕ, which controls the random removals in the operator "Neighbor graph removal". ϕ can take values from the set $\{1, 2, 3, 4, 5\}$. The higher the value ϕ is set to, the greater the chance that the nodes with the largest scores are selected. Experiments indicated that setting ϕ to 2 yields the best average results. Hence, the results shown in the remainder of the section were obtained by setting $\lambda = 0.1$, $\rho = 0.1$, and $\phi = 2$.

4.2 Contribution of each removal and insertion operator

The H-LNS heuristic utilizes five removal and five insertion operators. The usefulness of each of these ten operators is documented in Table 3, which shows the degree to which the average deviations from the lower bounds worsen when a particular operator is excluded from the algorithm. Experiments were conducted on the 24 instances in Set 1, with each instance run 20 times. The most useful removal operator was found to be the cluster removal operator, whereas "best insertion" constituted the most useful insertion operator. It was quite surprising that

[1] `http://project-osrm.org/`
[2] `https://feeds.citibikenyc.com/stations/stations.json`

Table 2: Parameters used in the algorithm

Parameter	Meaning
λ	A solution is passed to TS if it is within λ% of the best known solution x^*
ρ	Controls the amount of noise in "Time-based insertion with noise"
ϕ	Controls the randomness in "Neighborhood graph removal"
K	The number of consecutive iterations without improvement to x^*
γ	The number of TS iterations
q	The number of stations removed at each LNS iteration

excluding the regret insertion operator improved the average results. However, the results for the second set of instances did not improve when this operator was excluded.

Table 3: Statistics for the operators

Operator	% Degradation
Random removal 1	0.06
Random removal 2	0.01
Cluster removal	0.13
Radial ruin	0.07
Neighbor graph removal	0.04
Time-based insertion with noise	0.05
Best insertion	0.07
Biased random insertion	0.05
Regret insertion	-0.03
Random insertion	0.02

4.3 Comparison between CPLEX and H-LNS

The results were obtained by setting $\lambda = 0.1$, $\rho = 0.1$, $\phi = 2$, $K = 100$, $\gamma = 50$, and $q \in \{1, \lceil 0.1 \times |\mathcal{N}| \rceil\}$. An instance is denoted as X_Y_Z, where X denotes the number of stations in the instance, Y denotes the number of vehicles, and Z denotes the length of the planning horizon (i.e., s stands for short, $T = 9000$, and l stands for long, $T = 18000$). As this set of instances has only been applied by our method, we decided to compare the results obtained with the results obtained from CPLEX 12.4. CPLEX was set to run for a maximum of 2 hours, and both the lower and upper bounds are reported in Table 4 (if a feasible solution was obtained within the maximum running time of 2 hours; otherwise, a hyphen, "-", is written instead). CPLEX did not find an optimal solution for any of the 24 instances in this dataset. To demonstrate that H-LNS finds better solutions when more time is allocated to the search, Table 4 also compares the results between $K = 50$ and $K = 100$, showing that the average gap from the lower bounds for $K = 50$ is 5.12% using 23 seconds of computing time on average. The results are improved when K is increased to 100 (reduced to 4.90%), but at the expense of almost doubling the computing time required.

Table 4: Results for the first set of instances with $\alpha = 0$.

Instance	LB	UB	H-LNS[1] *Avg*	Gap[2]	H-LNS[1] *Best*	Gap[2]	CPU[3]	H-LNS[4] *Avg*	Gap[2]	H-LNS[4] *Best*	Gap[2]	CPU[3]
75_2_s	440.05	470.78	469.17	6.21	466.56	5.68	4.86	469.14	6.20	466.29	5.63	7.73
75_2_l	327.26	352.01	347.34	5.78	344.41	4.98	8.93	346.21	5.47	343.33	4.68	17.29
100_2_s	642.89	678.68	673.06	4.48	668.24	3.79	5.75	671.31	4.23	666.69	3.57	11.76
100_2_l	484.68	527.21	521.61	7.08	515.02	5.89	13.73	519.99	6.79	512.89	5.50	29.62
125_2_s	894.69	935.21	925.04	3.28	919.59	2.71	9.43	923.76	3.15	919.59	2.71	15.68
125_2_l	708.12	759.23	750.47	5.64	743.19	4.72	19.59	748.28	5.37	741.65	4.52	44.68
150_2_s	1115.64	1161.75	1149.06	2.91	1145.97	2.65	11.12	1147.41	2.77	1144.72	2.54	21.67
150_2_l	912.33	978.45	958.18	4.79	949.99	3.96	24.90	955.66	4.53	948.83	3.85	47.18
175_2_s	1270.85	-	1307.48	2.80	1302.99	2.47	12.98	1306.16	2.70	1302.94	2.46	23.84
175_2_l	1061.63	-	1111.23	4.46	1100.51	3.53	28.67	1109.75	4.34	1100.42	3.52	55.36
200_2_s	1491.33	-	1529.88	2.52	1526.09	2.28	16.74	1528.77	2.45	1525.00	2.21	27.15
200_2_l	1272.70	-	1323.04	3.80	1313.73	3.12	40.53	1322.78	3.79	1312.26	3.01	68.20
75_3_s	372.23	410.04	404.89	8.07	396.68	6.16	8.75	402.71	7.57	395.83	5.96	14.99
75_3_l	281.80	303.28	299.09	5.78	296.35	4.91	11.76	298.28	5.52	295.84	4.74	21.08
100_3_s	553.42	614.11	592.72	6.63	586.62	5.66	9.96	591.82	6.49	586.40	5.62	18.34
100_3_l	409.61	475.29	434.54	5.74	427.09	4.09	25.63	432.65	5.32	427.09	4.09	50.36
125_3_s	794.52	883.97	836.25	4.99	831.02	4.39	19.15	835.06	4.85	829.91	4.26	28.24
125_3_l	591.90	-	632.95	6.49	625.24	5.33	37.37	631.68	6.30	624.92	5.28	66.64
150_3_s	1008.36	-	1054.48	4.37	1048.48	3.83	22.71	1052.73	4.21	1046.85	3.68	43.96
150_3_l	766.10	-	818.78	6.43	808.64	5.26	40.18	817.15	6.25	805.87	4.94	76.85
175_3_s	1160.81	-	1213.04	4.31	1204.12	3.60	22.76	1207.95	3.90	1202.52	3.47	53.89
175_3_l	902.47	-	966.27	6.60	958.33	5.83	53.98	962.69	6.25	952.44	5.25	114.21
200_3_s	1377.33	-	1430.56	3.72	1422.41	3.17	27.25	1426.04	3.42	1419.94	3.00	57.51
200_3_l	1092.96	-	1161.36	5.89	1151.25	5.06	72.72	1159.51	5.74	1151.10	5.05	125.23
Average				5.12		4.30	22.89		4.90		4.15	43.39

[1] $K = 50$
[2] Deviation in % of the H-LNS results from the LB.
[3] CPU time (in seconds) to run the H-LNS.
[4] $K = 100$

4.4 Comparison among CPLEX, 3-step math heuristic, and H-LNS

Here, the results were obtained by setting $\lambda = 0.1$, $\rho = 0.1$, $\phi = 2$, $K = 50$, $\gamma = 50$, and $q \in \{1, \lceil 0.1 \times |\mathcal{N}| \rceil\}$. An instance is denoted as X_Y_Z, where X denotes the number of stations in the instance, Y denotes the number of vehicles, and Z denotes the workload level. Two sets of experiments were conducted with the second set of instances used by Forma et al. (2015). The results of the first set are presented in Table 5, with $\alpha = 0$. The H-LNS results are compared only with the CPLEX results, as Forma et al. (2015) did not conduct any experiments for $\alpha = 0$. CPLEX did not manage to find an optimal solution for any of the 30 instances within the 2-hour time limit. H-LNS obtained better feasible solutions than CPLEX, and also managed to reduce the optimality gap to 3.09% on average (with a minimal gap of 2.69%). The second set of experiments was run on the same set of instances, but with α set to 1/900. The 3-step math heuristic of Forma et al. (2015) was the only method used to conduct computational experiments on the second set of instances (with $\alpha = 1/900$). However, there is a slight difference between

their problem formulation and ours. In Forma et al. (2015), a station is not categorized as a pick-up or drop-off station. Hence, in theory, an optimal solution derived from their problem formulation could be better than that derived from ours. Nevertheless, comparing the results from the H-LNS and math heuristic is reasonable.

Table 6 shows the lower and upper bounds obtained from CPLEX (on our mathematical model), the results from the math heuristic, the results from the H-LNS, and the average deviations from our results to the lower bounds and math heuristic, respectively. The average computing time for running an instance is reported in the last column. Forma et al. (2015) did not specify the time taken to run each instance, although perusal of the text suggests that it took more than an hour to run, and their computer is faster than ours. For all 30 instances, our H-LNS managed to find better solutions than their math heuristic. The average improvement was 1.06%, and the maximum improvement was 1.48%. It is likely that greater improvement could be achieved by increasing K and/or γ, but at the expense of increasing the computing time. Currently, the average computing time is less than half a minute.

4.5 Comparison between LNS and H-LNS

The removal and insertion operators described in Sections 3.3 and 3.4 exhibit the effects of both diversification and intensification, although their intensification effect is not strong. Hence, it is necessary to apply a short TS to intensify the search in promising regions of the solution space. To demonstrate the effectiveness of including TS in the resulting H-LNS algorithm, computational experiments were performed on a pure large neighborhood search with TS excluded (i.e., lines 11-16 of Algorithm 1 were omitted).

Table 7 reports the results for the first set of instances obtained by running LNS with $K = 100$ (columns 2-3) and $K = 1000$ (columns 5-6). To allow a fair comparison between LNS and H-LNS, the average computing time between the two needed to be similar. Hence, we compared LNS (with $K = 1000$) with H-LNS (with $K = 50$ and $\gamma = 50$; see Table 4), and found H-LNS to obtain better solutions on all 24 instances. On average, the improvement was 0.79%, with LNS requiring slightly more computing time.

Experiments were also conducted with LNS on the second set of instances. Table 8 presents the LNS results with $K = 100$ (columns 2-3) and $K = 1000$ (columns 5-6). LNS (with $K = 1000$) was compared with H-LNS (with $K = 50$ and $\gamma = 50$; see Table 6) as the average computing times are similar. Again, H-LNS found better solutions than LNS for all 30 instances, with an average improvement of 0.67%.

Finally, the results of experiments conducted on the third set of instances are reported in Table 9. This table shows the results from LNS with $K = 1000$ (columns 2-3) and H-LNS with $K = 50$ and $\gamma = 50$ (columns 5-6). As with the previous experiments, H-LNS obtained better solutions than LNS for all 12 instances, with an average improvement of 0.49%.

5 Conclusion

This paper proposes a hybrid heuristic to solve the revised arc-index formulation of the multi-vehicle static repositioning problem in Raviv et al. (2013), where station characteristics are explicitly considered. The heuristic is based on large neighborhood search, but incorporates tabu search and various insertion and removal operators to improve the algorithmic performance. To illustrate the proposed heuristic's performance, it was tested on three sets of instances with up

to 518 stations and five vehicles. The results show our heuristic, hybrid large neighborhood search, to obtain better results than the 3-step math heuristic proposed by Forma et al. (2015). The average improvement over the math heuristic is 1.06%. In addition, the average computing time of our heuristic is less than half a minute, whereas that of the 3-step math heuristic is more than one hour. The computational results also confirm that our heuristic performs better when tabu search is incorporated, and that each insertion and removal operator contributes to solution accuracy. However, our heuristic cannot consider a situation in which it may be desirable to pick up or drop off bikes at balanced stations to help to balance neighboring stations with higher penalty costs. Therefore, a future research direction is to derive effective and efficient operators to handle the loading and unloading quantities at balanced stations.

Acknowledgments

This work was partially supported by a grant from the National Natural Science Foundation of China (71271183). This support is gratefully acknowledged. Thanks are also due to the two anonymous reviewers for their valuable comments.

A Repair procedure

The repair procedure is modified from Ho and Szeto (2014) for the multiple-vehicle case, and operates on an individual route. It starts by initializing all of the pick-up/drop-off quantities with zero, and then assigning the pick-up/drop-off quantity at each station with a minimal number of bikes to ensure that the solution becomes feasible. Then, one pick-up station and one drop-off station are selected, with their pick-up and drop-off quantities increased by at most two to maximize the reduction of the objective value. The problem of determining this pair of stations (p, d) in route v and the increase in pick-up/drop-off quantity φ is formulated as follows.

$$(p,d) = \underset{i\in W_v\cap\mathcal{P}, j\in W_v\cap\mathcal{D}}{\arg\max} \{f_i(s_i^0 - \sum_r y_{ir}^P) - f_i(s_i^0 - \sum_r y_{ir}^P - \varphi) + f_j(s_j^0 + \sum_r y_{jr}^D) - f_j(s_j^0 + \sum_r y_{jr}^D + \varphi) :$$
$$\varphi = \min\{s_i^0 - \sum_r y_{ir}^P, c_j - s_j^0 - \sum_r y_{jr}^D, \lfloor \tau/(U+L) \rfloor, 2\}\},$$

subject to capacity and loading constraints. Afterwards, the pick-up and drop-off quantities ($y_{pv}^P = y_{pv}^P + \varphi$ and $y_{dv}^D = y_{dv}^D + \varphi$) and τ are updated ($\tau = \tau - (U+L)\varphi$). The entire procedure is repeated until τ permits no further increase in the pick-up/drop-off quantity of any station or the objective value cannot be reduced.

References

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

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

Bent, R. and Van Hentenryck, P. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows, *Computers & Operations Research* **33**(4): 875–893.

Caggiani, L. and Ottomanelli, M. (2012). A modular soft computing based method for vehicles repositioning in bike-sharing systems, *Procedia - Social and Behavioral Sciences* **54**: 675–684.

Charon, I. and Hudry, O. (1993). The noising method: a new method for combinatorial optimization, *Operations Research Letters* **14**(3): 133–137.

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

Chow, J. Y. J. and Sayarshad, H. R. (2014). Symbiotic network design strategies in the presence of coexisting transportation networks, *Transportation Research Part B* **62**: 13–34.

Contardo, C., Morency, C. and Rousseau, L.-M. (2012). Balancing a dynamic public bike-sharing system, *Technical Report CIRRELT-2012-09*, Montréal.

de Chardon, C. M. and Caruso, G. (2015). Estimating bike-share trips using station level data, *Transportation Research Part B* **78**: 260–279.

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

Di Gaspero, L., Rendl, A. and Urli, T. (2013a). Constraint-based approaches for balancing bike sharing systems, *in* C. Schulte (ed.), *Principles and Practice of Constraint Programming*, Vol. 8124 of *Lecture Notes of Computer Science*, Springer, Berlin, pp. 758–773.

Di Gaspero, L., Rendl, A. and Urli, T. (2013b). A hybrid ACO+CP for balancing bicycle sharing systems, *in* M. J. Blesa Aguilera, C. Blum, P. Festa, A. Roli and M. Sampels (eds), *Hybrid Metaheuristics 8th International Workshop*, Vol. 7919 of *Lecture Notes of Computer Science*, Springer, Berlin, pp. 198–212.

Erdoğan, G., Battarra, M. and 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. and 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. and Tzur, M. (2015). A 3-step math heuristic for the static repositioning problem in bike-sharing systems, *Transportation Research Part B* **71**: 230–247.

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

Ho, S. C. and Szeto, W. Y. (2016). GRASP with path relinking for the selective pickup and delivery problem, *Expert Systems with Applications* **51**: 14–25.

Juan, A. A., Faulin, J., Ruiz, R., Barrios, B. and Caballé, S. (2010). The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem, *Applied Soft Computing* **10**(1): 215–224.

Kitthamkesorn, S., Chen, A., Xu, X. and Ryu, S. (2016). Modeling mode and route similarities in network equilibrium problem with go-green modes, *Networks and Spatial Economics* **16**(1): 33–60.

Kloimüllner, C., Papazek, P., Hu, B. and Raidl, G. R. (2014). Balancing bicycle sharing systems: An approach for the dynamic case, *in* C. Blum and G. Ochoa (eds), *Evolutionary Computation in Combinatorial Optimisation*, Vol. 8600 of *Lecture Notes in Computer Science*, Springer, Berlin, pp. 73–84.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem, *Proceedings of the American Mathematical Society* **7**(1): 48–50.

Lawson, A. R., Pakrashi, V., Ghosh, B. and Szeto, W. Y. (2013). Perception of safety of cyclists in Dublin city, *Accident Analysis & Prevention* **50**: 499–511.

Lin, J. H. and Chou, T. C. (2012). A geo-aware and VRP-based public bicycle redistribution system, *International Journal of Vehicular Technology* **2012**: 14 pages.

Lin, J.-R. and Yang, T.-H. (2011). Strategic design of public bicycle sharing systems with service level constraints, *Transportation Research Part E* **47**(2): 284–294.

Lin, J.-R., Yang, T.-H. and Chang, Y.-C. (2013). A hub location inventory model for bicycle sharing system design: Formulation and solution, *Computers & Industrial Engineering* **65**(1): 77–86.

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

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

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search, *Computers & Operations Research* **24**(11): 1097–1100.

Nair, R. and Miller-Hooks, E. (2011). Fleet management for vehicle sharing operations, *Transportation Science* **45**(4): 524–540.

Nair, R., Miller-Hooks, E., Hampshire, R. C. and Bušić, A. (2013). Large-scale vehicle sharing systems: Analysis of Vélib', *International Journal of Sustainable Transportation* **7**(1): 85–106.

Papazek, P., Kloimüllner, C., Hu, B. and Raidl, G. R. (2014). Balancing bicycle sharing systems: An analysis of path relinking and recombination within a GRASP hybrid, *in* T. Bartz-Beielstein, J. Branke, B. Filipič and J. Smith (eds), *Parallel Problem Solving from Nature – PPSN XIII*, Springer, Berlin, pp. 792–801.

Rainer-Harbach, M., Papazek, P., Raidl, G. R., Hu, B. and 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. and Kolka, O. (2013). Optimal inventory management of a bike-sharing station, *IIE Transactions* **45**(10): 1077–1093.

Raviv, T., Tzur, M. and 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.

Ropke, S. and Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, *Transportation Science* **40**(4): 455–472.

Ropke, S. and Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls, *European Journal of Operational Research* **171**(3): 750–775.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, *European Journal of Operational Research* **177**(3): 2033–2049.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H. and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle, *Journal of Computational Physics* **159**(2): 139–171.

Schuijbroek, J., Hampshire, R. and van Hoeve, W.-J. (2013). Inventory rebalancing and vehicle routing in bike sharing systems, *Technical report*, Carnegie Mellon University, USA.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems, *in* M. Maher and J.-F. Puget (eds), *Principles and Practice of Constraint Programming*, Vol. 1520 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 417–431.

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

Ting, C.-K. and Liao, X.-L. (2013). The selective pickup and delivery problem: Formulation and a memetic algorithm, *International Journal of Production Economics* **141**(1): 199–211.

Table 5: Results for the second set of instances with $\alpha = 0$.

Instance	LB	UB	H-LNS *Avg*	Gap[1]	H-LNS *Best*	Gap[1]	CPU[2]
75_2_light	489.78	501.75	493.29	0.71	492.56	0.57	7.37
75_2_real	489.78	494.20	490.58	0.16	490.37	0.12	8.78
75_2_heavy	552.35	595.80	588.40	6.13	582.95	5.25	8.52
75_3_light	489.78	501.24	491.00	0.25	490.42	0.13	9.38
75_3_real	489.78	492.12	489.80	0.00	489.78	0.00	8.19
75_3_heavy	508.87	565.10	537.86	5.39	534.27	4.75	10.08
100_2_light	650.51	676.32	667.90	2.60	666.31	2.37	14.69
100_2_real	647.89	670.48	653.73	0.89	652.36	0.69	12.21
100_2_heavy	759.30	825.60	806.64	5.87	800.04	5.09	11.55
100_3_light	647.71	-	654.64	1.06	652.93	0.80	17.97
100_3_real	647.71	-	649.09	0.21	648.88	0.18	15.88
100_3_heavy	695.16	-	738.79	5.90	732.71	5.12	20.62
125_2_light	817.68	895.01	845.41	3.28	843.38	3.05	16.28
125_2_real	810.52	848.83	825.38	1.80	822.97	1.51	16.97
125_2_heavy	1014.99	1145.56	1070.69	5.20	1064.13	4.62	14.29
125_3_light	809.21	-	827.41	2.20	824.91	1.90	28.12
125_3_real	809.10	-	814.49	0.66	812.64	0.44	27.90
125_3_heavy	914.10	-	990.55	7.72	980.79	6.80	22.53
150_2_light	998.88	-	1037.71	3.74	1034.79	3.47	25.99
150_2_real	989.81	-	1015.01	2.48	1011.89	2.18	18.45
150_2_heavy	1278.03	-	1336.01	4.34	1331.01	3.98	18.36
150_3_light	982.59	-	1012.99	3.00	1008.41	2.56	35.46
150_3_real	981.54	-	994.80	1.33	991.64	1.02	30.43
150_3_heavy	1154.59	-	1237.13	6.67	1227.66	5.95	30.56
200_2_light	1386.76	-	1433.25	3.24	1429.43	2.99	30.02
200_2_real	1369.52	-	1416.50	3.32	1412.28	3.03	22.62
200_2_heavy	1946.08	-	2004.01	2.89	1996.50	2.53	17.92
200_3_light	1352.34	-	1402.37	3.57	1395.83	3.12	50.27
200_3_real	1337.15	-	1379.33	3.06	1370.17	2.41	36.81
200_3_heavy	1792.09	-	1883.52	4.85	1869.91	4.16	30.80
Average				3.09		2.69	20.63

[1] Deviation in % of the H-LNS results from the LB.
[2] CPU time (in seconds) to run the H-LNS.

Table 6: Results for the second set of instances with $\alpha = 1/900$.

Instance	LB	UB	Math heuristic	H-LNS *Avg*	Gap[1]	Gap[2]	H-LNS *Best*	Gap[1]	Gap[2]	CPU[3]
75_2_light	502.01	522.24	519.95	515.04	2.53	-0.94	513.39	2.22	-1.26	12.03
75_2_real	500.37	515.82	511.19	510.97	2.07	-0.04	509.94	1.88	-0.24	12.00
75_2_heavy	563.70	634.90	616.49	604.20	6.70	-1.99	601.23	6.24	-2.48	9.57
75_3_light	501.68	523.49	519.95	515.41	2.66	-0.87	513.86	2.37	-1.17	15.27
75_3_real	500.14	522.41	511.88	510.88	2.10	-0.20	509.82	1.90	-0.40	14.68
75_3_heavy	527.69	594.85	576.22	562.94	6.26	-2.30	557.13	5.28	-3.31	10.23
100_2_light	668.77	703.74	696.21	690.78	3.19	-0.78	689.45	3.00	-0.97	15.59
100_2_real	661.23	682.41	679.43	675.04	2.05	-0.65	673.82	1.87	-0.83	14.91
100_2_heavy	771.45	865.90	825.72	822.42	6.20	-0.40	815.01	5.35	-1.30	11.80
100_3_light	668.26	-	697.11	691.79	3.40	-0.76	689.43	3.07	-1.10	23.84
100_3_real	661.11	686.49	677.11	675.66	2.15	-0.21	674.05	1.92	-0.45	19.32
100_3_heavy	715.73	-	788.17	761.82	6.05	-3.34	754.17	5.10	-4.31	23.12
125_2_light	834.60	-	873.81	867.14	3.75	-0.76	864.62	3.47	-1.05	17.90
125_2_real	825.52	871.43	851.26	846.72	2.50	-0.53	845.08	2.31	-0.73	16.60
125_2_heavy	1025.63	1141.70	1100.12	1086.27	5.58	-1.26	1079.37	4.98	-1.89	13.26
125_3_light	832.04	-	872.28	863.72	3.67	-0.98	861.37	3.41	-1.25	27.26
125_3_real	825.05	-	850.04	846.38	2.52	-0.43	844.12	2.26	-0.70	30.54
125_3_heavy	931.80	-	1026.84	1015.63	8.25	-1.09	1005.25	7.31	-2.10	25.56
150_2_light	1014.78	-	1071.10	1059.58	4.23	-1.08	1057.61	4.05	-1.26	23.42
150_2_real	1003.24	-	1034.66	1034.03	2.98	-0.06	1032.16	2.80	-0.24	19.44
150_2_heavy	1290.01	-	1380.39	1351.62	4.56	-2.08	1347.18	4.24	-2.41	18.29
150_3_light	1005.83	-	1055.03	1046.73	3.91	-0.79	1044.41	3.69	-1.01	37.80
150_3_real	1000.75	-	1033.40	1027.55	2.61	-0.57	1025.29	2.39	-0.78	26.85
150_3_heavy	1172.70	-	1287.05	1262.55	7.12	-1.90	1250.81	6.24	-2.82	27.73
200_2_light	1400.58	-	1467.71	1453.31	3.63	-0.98	1450.71	3.46	-1.16	24.44
200_2_real	1382.96	-	1452.10	1437.01	3.76	-1.04	1434.60	3.60	-1.21	21.71
200_2_heavy	1950.54	-	2043.04	2020.82	3.48	-1.09	2011.08	3.01	-1.56	15.88
200_3_light	1375.23	-	1456.80	1433.85	4.09	-1.58	1429.55	3.80	-1.87	50.90
200_3_real	1358.88	-	1428.88	1410.21	3.64	-1.31	1405.59	3.32	-1.63	35.37
200_3_heavy	1808.03	-	1942.73	1909.78	5.33	-1.70	1885.42	4.10	-2.95	34.37
Average					4.03	-1.06		3.62	-1.48	21.66

[1] Deviation in % of the H-LNS results from the LB.
[2] Improvement in % of the H-LNS results over the math heuristic.
[3] CPU time (in seconds) to run the H-LNS.

Table 7: Results for the first set of instances using LNS with $\alpha = 0$.

Instance	LNS[1] *Avg*	LNS[1] *Best*	CPU[2]	LNS[3] *Avg*	LNS[3] *Best*	CPU[2]
75_2_s	476.62	470.25	0.55	473.85	470.10	3.80
75_2_l	353.73	348.11	2.86	351.71	347.69	21.73
100_2_s	680.05	673.52	0.93	677.25	670.55	5.10
100_2_l	529.30	518.49	4.38	525.08	513.69	32.82
125_2_s	934.93	930.44	1.21	933.03	928.11	7.12
125_2_l	763.18	754.65	4.51	758.15	747.59	32.42
150_2_s	1156.83	1147.64	1.42	1152.77	1146.54	9.27
150_2_l	971.46	958.66	5.10	964.88	953.15	43.20
175_2_s	1315.71	1308.99	1.68	1311.32	1306.24	11.88
175_2_l	1126.96	1116.29	5.76	1120.64	1112.78	44.75
200_2_s	1535.81	1530.41	1.97	1533.72	1529.53	10.82
200_2_l	1337.68	1327.29	6.27	1332.31	1317.54	48.83
75_3_s	414.46	405.46	1.28	411.43	405.46	8.49
75_3_l	302.78	298.34	3.26	300.31	296.95	27.48
100_3_s	602.47	593.04	1.70	599.24	590.99	10.62
100_3_l	444.91	438.31	4.61	439.16	432.28	52.65
125_3_s	855.84	839.59	1.93	849.10	837.75	15.00
125_3_l	647.96	637.19	6.23	641.96	636.62	55.08
150_3_s	1066.66	1059.81	2.65	1060.60	1054.03	17.79
150_3_l	835.22	826.22	9.46	828.51	821.56	82.29
175_3_s	1222.95	1209.18	3.76	1215.78	1207.42	22.63
175_3_l	979.14	963.36	9.30	971.56	959.80	94.17
200_3_s	1444.49	1430.34	3.59	1434.09	1426.39	28.06
200_3_l	1179.04	1165.95	10.29	1169.61	1157.09	98.64
Average			3.95			32.69

[1] $K = 100$
[2] CPU time (in seconds) to run the LNS.
[3] $K = 1000$

Table 8: Results for the second set of instances using LNS with $\alpha = 1/900$.

Instance	LNS[1] *Avg*	LNS[1] *Best*	CPU[2]	LNS[3] *Avg*	LNS[3] *Best*	CPU[2]
75_2_light	519.87	516.24	1.01	518.78	515.06	8.26
75_2_real	513.04	511.18	1.33	512.08	510.37	14.57
75_2_heavy	618.55	609.26	1.61	612.51	605.87	20.03
75_3_light	521.46	518.23	1.11	520.27	515.53	8.14
75_3_real	514.65	512.17	1.59	513.21	510.80	11.28
75_3_heavy	574.55	569.05	1.55	570.26	560.68	14.79
100_2_light	696.14	691.23	1.61	694.86	690.93	11.30
100_2_real	677.61	674.94	1.91	676.46	674.90	12.34
100_2_heavy	836.46	820.94	1.91	830.58	819.54	14.98
100_3_light	697.13	692.85	1.54	695.22	692.71	11.41
100_3_real	680.04	676.65	1.47	678.14	674.60	10.90
100_3_heavy	784.24	767.28	3.02	773.65	758.92	31.61
125_2_light	873.25	867.17	1.99	870.89	866.46	16.92
125_2_real	850.59	847.52	1.90	849.17	846.39	16.33
125_2_heavy	1107.05	1089.16	1.95	1100.67	1085.62	13.23
125_3_light	869.99	864.72	2.49	867.67	862.76	23.48
125_3_real	851.94	848.27	2.48	848.74	846.59	23.62
125_3_heavy	1046.59	1020.48	3.07	1032.21	1016.98	29.41
150_2_light	1068.72	1060.59	2.39	1064.76	1060.22	21.81
150_2_real	1040.32	1035.85	2.05	1037.70	1034.53	19.37
150_2_heavy	1379.66	1349.78	2.14	1369.52	1348.60	15.04
150_3_light	1054.52	1047.28	3.49	1051.52	1045.60	31.98
150_3_real	1032.21	1028.78	3.43	1030.33	1026.52	30.57
150_3_heavy	1280.18	1264.78	4.22	1272.75	1258.36	27.12
200_2_light	1462.34	1453.93	2.99	1457.87	1452.41	29.77
200_2_real	1446.30	1437.78	2.54	1442.98	1437.57	17.75
200_2_heavy	2039.69	2029.61	2.30	2031.92	2019.38	16.03
200_3_light	1445.89	1436.34	4.41	1442.93	1431.99	38.44
200_3_real	1417.40	1411.31	3.80	1414.65	1409.64	32.35
200_3_heavy	1936.43	1915.63	3.76	1924.35	1908.59	29.09
Average			2.37			20.06

[1] $K = 100$
[2] CPU time (in seconds) to run the LNS.
[3] $K = 1000$

Table 9: Results for the third set of instances using LNS and H-LNS with $\alpha = 0$.

Instance	LNS[1] *Avg*	LNS[1] *Best*	CPU[2]	H-LNS[3] *Avg*	H-LNS[3] *Best*	CPU[2]
302_3	2399.32	2388.86	110.49	2388.50	2383.12	87.81
302_4	2345.54	2339.01	125.29	2335.51	2326.23	114.04
302_5	2300.32	2288.67	146.18	2288.59	2278.89	162.34
349_3	2545.47	2539.75	127.60	2536.83	2531.42	100.16
349_4	2490.62	2484.89	130.57	2478.19	2471.63	143.49
349_5	2438.82	2429.16	170.52	2427.88	2422.53	188.28
471_3	2687.49	2672.92	166.61	2669.94	2664.42	205.47
471_4	2627.11	2615.79	221.18	2614.49	2608.27	210.32
471_5	2578.14	2569.45	261.61	2566.67	2559.23	297.80
518_3	2836.49	2823.28	155.77	2817.17	2811.95	201.30
518_4	2770.36	2762.75	232.62	2757.68	2750.67	221.50
518_5	2716.89	2708.16	292.05	2704.72	2696.44	286.98
Average			178.37			184.96

[1] $K = 1000$
[2] CPU time (in seconds) to run the heuristic.
[3] $K = 50$