Transportation Research Part C 92 (2018) 208–242

Contents lists available at ScienceDirect

Transportation Research Part C

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

A static free-floating bike repositioning problem with multiple heterogeneous vehicles, multiple depots, and multiple visits

Ying Liu[a,b], W.Y. Szeto[a,b,*], Sin C. Ho[a,b,c]

[a] Department of Civil Engineering, The University of Hong Kong, Hong Kong
[b] The University of Hong Kong Shenzhen Institute of Research and Innovation, Shenzhen, China
[c] Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
Free-floating bike sharing
Static bike repositioning problem
Multiple heterogeneous vehicles
Multiple depots
Multiple visits

ABSTRACT

In this paper, a bike repositioning problem with multiple depots, multiple visits, and multiple heterogeneous vehicles for the free-floating bike-sharing system (FFBSS) is studied. Two types of nodes (i.e., easily and hardly access nodes) with different penalties are defined to represent different convenience levels of getting bikes from the FFBSS. The objective of the repositioning is to minimize the weighted sum of the inconvenience level of getting bikes from the system and the total unmet demand and the total operational time. To solve this problem, an enhanced version of chemical reaction optimization (CRO) is developed. A loading and unloading quantity adjustment procedure with the consideration of the node characteristics, including the type of node and its current state (i.e., in a balanced, surplus, or deficit state) is proposed and incorporated into this version to improve its solution quality. A concept of the nearby-node set is also proposed to narrow the search space. Numerical results are presented and indicate that compared to the traditional CRO and CPLEX, the enhanced CRO improves solution quality and has potential to tackle the repositioning problem for larger, longer repositioning duration, and more vehicle instances. The results also demonstrate the effectiveness of the proposed adjustment procedure.

1. Introduction

A bike-sharing system (BSS) is a short rental service to provide customers with bikes for shared use. It has developed rapidly and worldwide. As of 28 November 2017, public bike-sharing systems were available in about 1488 cities and included approximately 18,740,100 bikes around the world (Meddin and DeMaio, 2017).

There are currently two types of BSSs operated worldwide: traditional BSS and free-floating BSS (FFBSS). In a traditional BSS, users have to rent bikes from the designated docking stations and return them to the available lockers in the docking stations after use. In some FFBSSs, bike racks or any solid frame or standalone can be used to lock bikes instead of docking stations, which are also the most costly component in a traditional BSS. The bike rack cost is low and hence the setup cost is lower than that of a traditional BSS and the number of racks in an FFBSS can be very large. Some FFBSSs (e.g., the Mobike system, China) removed the concept of bike racks or related concept. The locks in those systems simply immobilize the rear wheels and a smartphone provides the user interface for locating, checking out, returning, locking, and payment. With the help of global positioning system (GPS), users can reserve or directly rent the nearest available bike and return it almost anywhere in the operating area at the end of the trip. With this feature, users of FFBSS do not need to spend time on searching for an available locker at a docking station to return a bike. This feature makes an FFBSS more flexible and user-friendly,

* Corresponding author at: Department of Civil Engineering, The University of Hong Kong, Hong Kong and The University of Hong Kong Shenzhen Institute of Research and Innovation, Shenzhen, China.
E-mail address: ceszeto@hku.hk (W.Y. Szeto).

https://doi.org/10.1016/j.trc.2018.02.008
Received 4 June 2017; Received in revised form 28 November 2017; Accepted 9 February 2018
Available online 10 May 2018

compared to a traditional BSS. However, it also results in a more dispersed distribution of the bikes and raises a problem that some bikes are parked at the locations uneasy to observe by the public (Abdullah, 2017). Users find difficulties in searching for an available bike based on its location provided by GPS as the system continues to operate (Abdullah, 2017). Same as a traditional BSS, reallocating the bikes in an FFBSS is a necessity to improve the system performance, such as the convenience of users in getting bikes from the sharing system and the demand dissatisfaction. This relocation problem is called a bike-sharing repositioning problem (BRP).

The BRPs for traditional BSSs have been examined and solved by many existing studies. Among them, very few studies (Caggiani and Ottomanelli, 2012; Contardo et al., 2012; Regue and Recker, 2014; Brinkmann et al., 2015a, 2015b; Labadi et al., 2015; Zhang et al., 2017; Shui and Szeto, 2018) deal with the dynamic bike repositioning problem that captures the demand variation over the repositioning period. Most existing studies (Benchimol et al., 2011; Nair and Miller-Hooks, 2011; Chemla et al., 2013; Di Gaspero et al., 2013a, 2013b, 2016; Nair et al., 2013; Papazek et al., 2013, 2014; Raviv et al., 2013; Rainer-Harbach et al., 2013, 2015; Dell'Amico et al., 2014, 2016; Erdoğan et al., 2014, 2015; Ho and Szeto, 2014, 2017; Brinkmann et al., 2015a, 2015b; Forma et al., 2015; Alvarez-Valdes et al., 2016; Casazza, 2016; Kadri et al., 2016; Li et al., 2016; Szeto et al., 2016; Cruz et al., 2017; Schuijbroek et al., 2017) focus on the static bike repositioning problem in which it is commonly assumed that there is no or negligible demand during the repositioning operation and the objective is to arrange bikes in the system for the next working day. To solve the real-life BRPs, most of these existing studies developed inexact solution methods, including approximation methods, heuristics, metaheuristics, and hybrid heuristic and exact methods.

To have a good relocation strategy, accurate estimation of bike demand is necessary. Some effort focused on bike demand forecasting. For example, Rudloff and Lackner (2014) developed demand models for bikes and available lockers, which can predict the times when stations tend to full or empty over the course of the coming week. Singhvi et al. (2015) predicted the bike usage pattern of Citi Bike in New York during morning peak hours and bike demand by considering taxi usage, weather, and spatial variables. However, these studies focus on traditional BSSs.

Compared to the studies for traditional BSSs, very few papers focused FFBSSs, including forecasting demand for FFBSSs and free-floating BRPs (FFBRPs). For example, to enhance the bike reallocation, Reiss and Bogenberger (2016) identified the mobility patterns of the bike usage and forecasted the upcoming demand, while Caggiani et al. (2016) proposed a method to generate spatiotemporal clusters and forecasted the bike use trend.

As the time of this writing, only one study deals with an FFBRP. Pal and Zhang (2017) considered a multiple-vehicle static repositioning for the FFBSS. The objectives are to make the FFBSS at its optimal state (i.e., no stations are in a deficit state) and minimize the makespan of the vehicles. Multiple-visits to a station with monotone loading or unloading operation are allowed. Their formulation is only modified from the formulation proposed by Erdoğan et al. (2015) for handling the multiple-vehicle repositioning as if a traditional BRP. To solve this problem, a hybrid nested large neighborhood search with variable neighborhood descent algorithm is proposed. However, they do not consider any unique feature of an FFBSS such as the inconvenience level of getting bikes when formulating and solving the problem. According to Abdullah (2017), it is sometimes difficult for bike users to find unused bikes in an FFBSS as they are put in inconvenience, difficult to see, or uncommon locations. The utilization rates of those bikes are relatively low, which does not allow the demand dissatisfaction in the system to be fully minimized. To improve the utilization rate and demand dissatisfaction, a new methodology is needed to model this unique feature and determine a repositioning strategy.

In this study, we consider the convenience level of getting bikes from the FFBSS. Two types of nodes are defined based on the convenience level: Easily accessed nodes represent the locations well-known to the public (e.g., the designated locations of bike racks or popular locations), whereas hardly accessed nodes are defined as the locations not popular to the public, not easy to be found, or introducing long search time to the public (e.g., the locations of dispersed bikes). In this FFBRP, bikes at hardly accessed nodes are transported to easily accessed nodes, especially to those in deficit. Thus, reallocating bikes for this FFBSS is not only to reduce the total unmet demand, but also to increase the convenience level of getting bikes from the system. This relocation is assumed to be performed in the night time by multiple vehicles and hence the problem is formulated as a multiple-vehicle static FFBRP. To solve this BRP, an enhanced version of chemical reaction optimization (CRO) is proposed. Compared with the traditional version, the enhanced version has four major differences: First, the enhanced CRO considers the characteristics of the FFBRP, i.e., two types of nodes. Second, the enhanced CRO applies a newly proposed operator to adjust loading/unloading quantities, especially at hardly accessed nodes, to improve its solution quality. Third, a solution adjustment strategy is adopted in the enhanced version to ensure the feasibility and good quality of the solutions. Fourth, the enhanced version uses different neighborhood operators and criteria to obtain new routes. To demonstrate the effectiveness of the enhanced CRO, different test scenarios were used and the results obtained from IBM ILOG CPLEX, the original CRO, and the enhanced CRO were compared. In addition, the performance of the proposed subroutine is discussed.

The contributions of this paper are as follows:

- This paper formulates an FFBRP with the consideration of the convenience level, in which this special feature associated with an FFBSS has not been modeled before.
- This problem considers (1) multiple depots; (2) multiple heterogeneous vehicles; (3) fixed starting depot and flexible ending depot for each vehicle; (4) multiple-visits of nodes by the same vehicle, and (5) non-monotone loading and unloading operations at nodes, which is more complicated than existing traditional BRPs.
- An enhanced version of CRO is proposed. It considers the node characteristics in developing a subroutine to adjust the loading or unloading quantities at nodes, especially for hardly accessed nodes. These quantities are adjusted based on the type of node and its current state (i.e., in a balanced, surplus, or deficit state). The results from our experiments show that this subroutine contributes to better solutions.
- This study improves the traditional CRO to solve the FFBRP. A concept of the nearby-node set is proposed to narrow the search space. The results indicate that the enhanced CRO obtains better solutions than the traditional CRO.

2. Formulation

The free-floating bike-sharing system (FFBSS) is represented by a complete directed graph $G_0 = (V_0, A_0)$, where V_0 and A_0 are sets of nodes and arcs, respectively. In this study, multiple depots are considered. Vehicles may not start and end the repositioning at the same depot. All nodes can be visited multiple times and treated as *buffer nodes*. That is, the non-monotone loading and unloading operations at each of these nodes are allowed. We also assume that the demand at hardly accessed nodes is 0. The repositioning duration is discretized into many equal intervals called periods. The repositioning problem is formulated as a multiple period problem using the following notations.

Sets

$\overline{D}$	A set of depots.
V_E	A set of nodes which are easily accessed by customers (e.g., bike racks/stations in the FFBSS).
V_H	A set of nodes which are hardly accessed by customers (e.g., dispersed bikes in the FFBSS).
V	A set of nodes, including easily and hardly accessed nodes, i.e., $V = V_E \cup V_H$.
V_0	A set of nodes, including depots, and easily and hardly accessible nodes, i.e., $V_0 = \overline{D} \cup V_E \cup V_H$.
K	A fleet of heterogeneous vehicles $K = \{1, 2, ..., \|K\|\}$.

Parameters

s_i^0	The initial inventory level at node $i \in V_0$.
q_i	The optimal inventory level at node $i \in V$.
a_{ik}	The parameter indicating the starting depot of each vehicle $k \in K$; If $a_{ik} = 1$, vehicle k starts the repositioning operation at depot $i \in \overline{D}$, 0 otherwise.
cP	The penalty factor of convenience per bike assigned to each node $i \in V_H$. It is used to measure the convenience of getting bikes from the FFBSS.
Q_k	The capacity of vehicle k.
τ	The length of a short period (in s).
T	The repositioning duration (in s).
t_{ij}	Travel time from node i to node j (in s).
L	The time required to load a bike onto the vehicle.
U	The time required to unload a bike from the vehicle.
T'	The number of periods in the repositioning duration, i.e., $T' = \lfloor T/\tau \rfloor$.
t'_{ij}	Discretized travel time between nodes, i.e., $t'_{ij} = \begin{cases} \lceil t_{ij}/\tau \rceil & i \neq j, i, j \in V_0 \\ 1 & i = j, i, j \in V_0 \end{cases}$.

Decision variables

x_{ijtk}	A binary variable that defines the route of vehicle k. If $x_{ijtk} = 1$, vehicle k departs from node i to node j during period t; 0 otherwise.
s_{it}	The inventory level at node $i \in V_0$ at the end of period t, $t = 1, ..., T'$.
y_{itk}^L	The number of bikes picked up at node i by vehicle k during period t.
y_{itk}^U	The number of bikes dropped off at node i by vehicle k during period t.
f_{ijtk}	The number of bikes on vehicle k when traveling from node i to node j during period t.
ψ_i	The auxiliary variable associated with node $i \in V_E$ used to obtain the unmet demand.

The repositioning problem is formulated as follows.

$$\min z = \sum_{i\in V_E} \psi_i + \sum_{i\in V_H} cP\cdot s_{iT'} + \mu\cdot \sum_{k\in K}\left(\sum_{i,j\in V_0}\sum_{t\geqslant t'_{ij}} t_{ij}\cdot x_{i,j,t-t'_{ij},k} + \sum_{i\in V_0}\sum_{t\geqslant 1}(L\cdot y^L_{itk} + U\cdot y^U_{itk})\right) \tag{1}$$

s.t.

$$\psi_i \geqslant q_i - s_{iT'}, \quad \forall\, i \in V_E \tag{2}$$

$$s_{i0} = s^0_i, \quad \forall\, i \in V_0 \tag{3}$$

$$s_{it} = s_{i,t-1} + \sum_{k\in K}(y^U_{itv} - y^L_{itv}), \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T' \tag{4}$$

$$\sum_{j\in V_0} x_{ij0k} = a_{ik}, \quad \forall\, i \in \overline{D}, \quad \forall\, k \in K \tag{5}$$

$$\sum_{j\in V_0} x_{ij0k} = 0, \quad \forall\, i \in V, \quad \forall\, k \in K \tag{6}$$

$$\sum_{i\in \overline{D}}\sum_{j\in V_0} x_{j,i,T'-t'_{ji},k} = 1, \quad \forall\, k \in K \tag{7}$$

$$\sum_{j\in V_0, t\geqslant t'_{ji}} x_{j,i,t-t'_{ji},k} = \sum_{h\in V_0} x_{ihtk}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{8}$$

$$\sum_{j\in V_0, t\geqslant t'_{ji}} f_{j,i,t-t'_{ji},k} = \sum_{h\in V_0} f_{ihtk} + y^U_{itk} - y^L_{itk}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{9}$$

$$f_{ijtk} \leqslant Q_k\cdot x_{ijtk}, \quad \forall\, i, j \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{10}$$

$$f_{ij0k} = 0, \quad \forall\, i, j \in V_0, \quad \forall\, k \in K \tag{11}$$

$$f_{ijT'k} = 0, \quad \forall\, i, j \in V_0, \quad \forall\, k \in K \tag{12}$$

$$y^L_{itk} \leqslant \min(2\overline{L}, Q_k)\cdot \sum_{j\in V_0} x_{ijtk}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{13}$$

$$y^U_{itk} \leqslant \min(2\overline{U}, Q_k)\cdot \sum_{j\in V_0} x_{ijtk}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{14}$$

$$\sum_{i,j\in V_0}\sum_{t\leqslant w, t\geqslant t'_{ij}} t_{ij}\cdot x_{i,j,t-t'_{ij},k} + \sum_{i\in V_0}\sum_{t\leqslant w}(L\cdot y^L_{itk} + U\cdot y^U_{itk}) \leqslant w\cdot\tau, \quad \forall\, w = 1, \ldots, T', \quad \forall\, k \in K \tag{15}$$

$$\sum_{i,j\in V_0}\sum_{t\leqslant w} t_{ij}\cdot x_{ijtk} + \sum_{i\in V_0}\sum_{t\leqslant w}(L\cdot y^L_{itk} + U\cdot y^U_{itk}) \geqslant (w-2)\cdot\tau, \quad \forall\, w = 1, \ldots, T', \quad \forall\, k \in K \tag{16}$$

$$x_{ijtk} \in \{0, 1\}, \quad \forall\, i, j \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{17}$$

$$y^L_{itk} \geqslant 0, \ \text{integer}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{18}$$

$$y^U_{itk} \geqslant 0, \ \text{integer}, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{19}$$

$$s_{it} \geqslant 0, \quad \forall\, i \in V_0, \quad \forall\, t = 1, \ldots, T' \tag{20}$$

$$f_{ijtk} \geqslant 0, \quad \forall\, i, j \in V_0, \quad \forall\, t = 1, \ldots, T', \quad \forall\, k \in K \tag{21}$$

$$\psi_i \geqslant 0, \quad \forall\, i \in V_E \tag{22}$$

The objective (1) aims to minimize the sum of following: (1) the total number of the unsatisfied customers (or total unmet demand) reflected by $\sum_{i\in V_E}\psi_i$, (2) the inconvenience of getting a bike from the FFBSS, which is reflected by the total penalties for bikes parked at hardly accessed nodes calculated by $\sum_{i\in V_H} cP\cdot s_{iT'}$, and (3) weighted vehicles' total operational time, including the travel times between nodes and the service time for loading and unloading operations. Note that $\sum_{i\in V_E}\psi_i$ is equivalent to $\sum_{i\in V_E}\psi^*_i$, where ψ^*_i is the minimum value of ψ_i, because the objective function is minimized. ψ^*_i represents the unmet demand at node i at the end of the repositioning operation, i.e., $\psi^*_i = \max(q_i - s_{iT'}, 0)$. Constraint (2) defines ψ_i.

Constraints (3) and (4) are inventory related constraints. Constraint (3) defines the initial inventory at each node. Constraint (4) depicts the conservation of bikes at each node during period t; if $t = T'$, it defines the inventory level at each node at the end of the repositioning operation.

Constraints (5)–(8) are used to define vehicle routes. Constraint (5) limits that each vehicle must start the repositioning operation at its starting depot. Constraint (6) restricts that each vehicle cannot start the repositioning operation at any node other than its starting depot. Constraint (7) indicates that each vehicle must return to a depot $i \in \overline{D}$ (which may be different from its starting depot) at the end of the repositioning operation. Constraint (8) means that if vehicle k arrives at node i during period t, it must depart from node j to node i during period $t-t'_{ij}$. Constraints (5)–(8) imply that each node including depots, easily accessed nodes, and hardly accessed nodes can be visited multiple times by a single vehicle.

Constraints (9)–(12) are related to bike loads on vehicles. Constraint (9) states the conservation of bicycles on vehicle k in period t. Constraint (10) is the vehicle capacity constraint. It ensures that the number of bikes on vehicle k should not exceed its capacity. Constraint (11) indicates the initial state of vehicles to be empty before the repositioning operation starts ($t = 0$). (However, bikes can be loaded onto vehicles at depots when $t > 0$.) Constraint (12) requires that there is no bike on each vehicle after the operation ends ($t = T'$). It implies that all of the bikes loaded onto the vehicles are unloaded at the end of the repositioning operation.

Constraints (13) and (14) are used to restrict loading and unloading quantities. Constraint (13) ensures that the loading quantity at each node i within each period t should not exceed both twice the actual maximum number (i.e., $\overline{L}$ and $\overline{U}$) and vehicle capacity if node i is visited by vehicle k in period t. Similarly, constraint (14) confines the unloading quantities at node $i \in \overline{D} \cup V_E$. Constraints (13) and (14) imply that the temporary storage/supply at any node is allowed. Using twice the actual maximum number enables the utilization of the time slack that may have been created by actual (rather than rounded up) travel time.

Constraints (15) and (16) define the upper and lower bounds of the total operational time, including route travel time and service time for loading and unloading operations, for each period and vehicle. The actual travel time t_{ij} (which can be a real number) is used on the left-hand sides of these constraints. Constraint (15) implies that the repositioning operation carried out by each vehicle must finish within the repositioning duration for the case $w = T'$. Constraint (16) is used to keep the time of the planned schedule close to its execution (Raviv et al., 2013).

Constraints (17)–(22) are the domain constraints.

Unlike the classical arc-indexed formulation, the formulation stated above can capture multiple visits to a node by a single vehicle and the synchronization of vehicles and is modified from the time-indexed formulation proposed by Raviv et al. (2013). The main modifications are stated as follows:

(i) A different objective function that considers total unmet demand, total penalty for bikes at hardly accessed nodes and total operational time (including travel time and loading/unloading time) is used (refer to objective (1), and constraints (2) and (22)).
(ii) In our problem, multiple-depot and multiple-vehicle are considered. Vehicles start the repositioning operation from their starting depot where they are parked (They may not be parked at the same depot). Constraint (5) is added to define the starting depot for each vehicle.
(iii) It is assumed that each vehicle can return to a different depot from the one it starts the repositioning operation. Constraint (7) is added to define that the vehicle must return to any depot at the end of the repositioning operation.
(iv) It is assumed that all vehicles are empty before and after the repositioning operation (refer to constraints (11) and (12)).
(v) In this problem, it is assumed that the bikes can be parked anywhere in the sharing system. Therefore, the station capacity constraint is removed.
(vi) When $x_{j,i,t-t'_{ji},k}$ is used in constraint (15), we must ensure that $t-t'_{ji} \geqslant 0$ so that $x_{j,i,t-t'_{ji},k}$ is well-defined.
(vii) When $x_{j,i,t-t'_{ji},k}$ and $t-t'_{ji} \geqslant 0$ were used in defining constraint (16), extra loading and unloading of bikes at a station could occur during $t-t'_{ji} < 0$ because these extra operations were used to induce operational time to ensure that the lower bound requirement is satisfied. Therefore, in the revised constraint, x_{ijtk} is used instead to avoid such problems.

Fig. 1 illustrates that our problem allows multiple visits to a node by a single vehicle and the synchronization of different vehicles to reduce the unmet demand and total penalty as much as possible within a time limit. Moreover, it illustrates that the starting depot

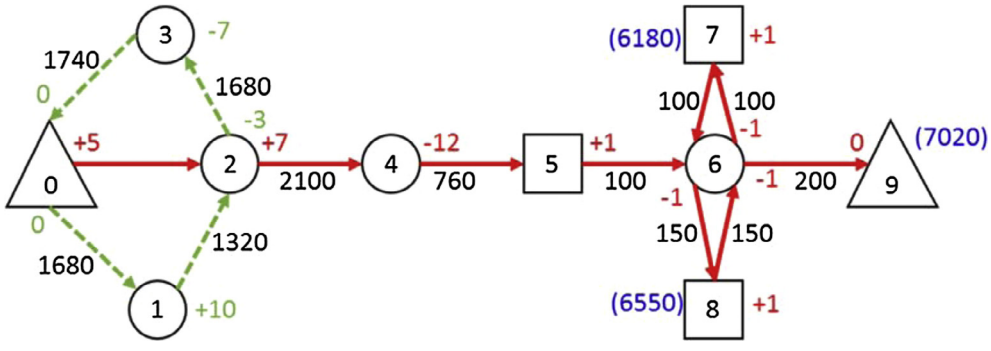

Fig. 1. Problem illustration.

Table 1
The properties of a molecule and their corresponding mathematical meaning.

Notation	Chemical meaning	Mathematical meaning in the BRP
ω	Molecular structure	A solution
PE	Potential energy	The objective function value of a solution
KE	Kinetic energy	The tolerance of having a worse solution than the current solution

of a vehicle can be different from the ending depot. Triangle nodes represent depots, circle nodes represent easily accessed nodes, and square nodes represent hardly accessed nodes. The numbers inside those nodes are node numbers. The number not in brackets next to each node represents the loading or unloading quantity; a positive sign means loading and a negative sign means unloading. The numbers in brackets next to some nodes are vehicle's arrival times at those nodes. For clarity, not all arrival times are presented. The number next to each arc is the travel time in seconds. The repositioning duration is 7200 s. The loading or unloading time for a bike is 60 s. There are two vehicles. Vehicle 1 has a capacity of 10 and vehicle 2 has a capacity of 20. The route of vehicle 1 (in dashed lines) starts and ends at the same depot while the other one (in solid lines) does not. Both vehicles are initially empty and start their operation at the same time. Vehicle 1 leaves node 0 at time zero while vehicle 2 picks up 5 bikes at node 0 at time zero and leaves node 0 at 300 s. At node 2, 4 extra bikes can be provided to serve other nodes without introducing demand dissatisfaction there but vehicle 2 picks up 3 more bikes there to serve node 4 (the largest demand station), which requires 12 bikes to reach a balanced state Node 2 is then supplied 3 bikes by vehicle 1 after vehicle 2 leaves node 2. Node 2 is, therefore, a temporary supply node. Vehicle 2 also visits node 6 three times to pick up all the bikes at (adjacent) hardly accessed nodes to minimize the total penalty. In this example, it can be easily checked that the vehicle capacity constraint is satisfied for each vehicle throughout the repositioning operation and the operational time of each of the two vehicles is less than 7200 s.

3. Enhanced chemical reaction optimization

CRO loosely mimics what happens to molecules in a chemical reaction system microscopically (Lam and Li, 2010). It tries to capture the phenomenon that reactions give products with the lowest energy. CRO has two major components: molecules and elementary reactions. Each molecule, denoted by M, is characterized by the properties listed in Table 1.

Elementary reactions mimic the occurrence of sequences of collisions among molecules to get their lowest energy state. Collisions under different conditions lead to different elementary reactions, each of which may have a different way of manipulating the energies of the involved molecule(s). In CRO, there are four types of elementary reactions: (1) on-wall ineffective collision, (2) decomposition, (3) intermolecular ineffective collision, and (4) synthesis. The operators used in these reactions guide the way of obtaining resultant solutions from reactant solutions. The classical reaction operators can be referred to Lam and Li (2010).

In this paper, an enhanced version of CRO is proposed to solve the multi-vehicle FFBRP and has the same algorithmic framework as the original CRO (see Fig. 2). The remainder of this section will describe the algorithmic details and further discuss the differences of the implementation details between the original and enhanced CRO.

3.1. Solution representation

A solution ω is represented by $\omega = (r, O)$, where r and O are vectors to store vehicle routes and the loading/unloading quantity at each visited node. Let n_k be the total number of nodes (excluding the starting and ending depots) visited by vehicle k, where $k = 1, 2...,|K|$. The nodes visited by vehicle k are denoted as $r^h \in V_0$, $h = \sum_{k'=1}^{k-1}(n_{k'} + 2) + l$, $l = 0, 1, ..., n_k + 1$ and hence $r = (r^h)$. Vehicles are assumed to start and end repositioning at depots. The depots are therefore the first and last elements in the vehicle routes, (i.e., $r^h \in \overline{D}$, where $h = \sum_{k'=1}^{k-1}(n_{k'} + 2)$ or $h = \sum_{k'=1}^{k-1}(n_{k'} + 2) + n_k + 1$). For illustrative purposes, in this section, two depots are considered, denoted by 0 and 1; stations are numbered from 2 onwards. However, the generalization of this representation to multiple depot scenarios is straightforward.

The elements of O are defined as $O^h = \overline{y}_h^L - \overline{y}_h^U$, where $\overline{y}_h^L$ and $\overline{y}_h^U$ denote the loading and unloading quantities at node r^h, respectively. A positive integer represents a loading operation, while a negative integer indicates an unloading operation.

Note that the formulation introduced in Section 2 is time-based, whereas the solution representation in our CRO is sequence-based. Therefore, instead of the time index t, we use h to indicate the position of a node visited by a vehicle in a solution, which is used to deduce the order of visits. Based on the order, the arrival time at each node can be computed. Hence, for notational convenience, the time and vehicle indices, i.e., t and k, are sometimes omitted.

A feasible solution for the repositioning operation performed by two vehicles with the capacities of 10 and 20 bikes is shown in Fig. 3. In this example, $n_1 = 6$ and $n_2 = 5$; vehicle 1 starts and ends the repositioning at depot 0, while vehicle 2 starts at depot 1 at the same time as vehicle 1 and returns to depot 0 at the end of the repositioning.

3.2. Nearby-node set

To narrow the search space and speed up the search algorithm, the concept of the nearby-node set is proposed, which considers the problem's characteristics. Define $\overline{Q}_\varepsilon$ as the capacity of vehicle type $\varepsilon \in \{1, ..., \overline{K}\}$, where $\overline{K}$ is the number of vehicle types. For each

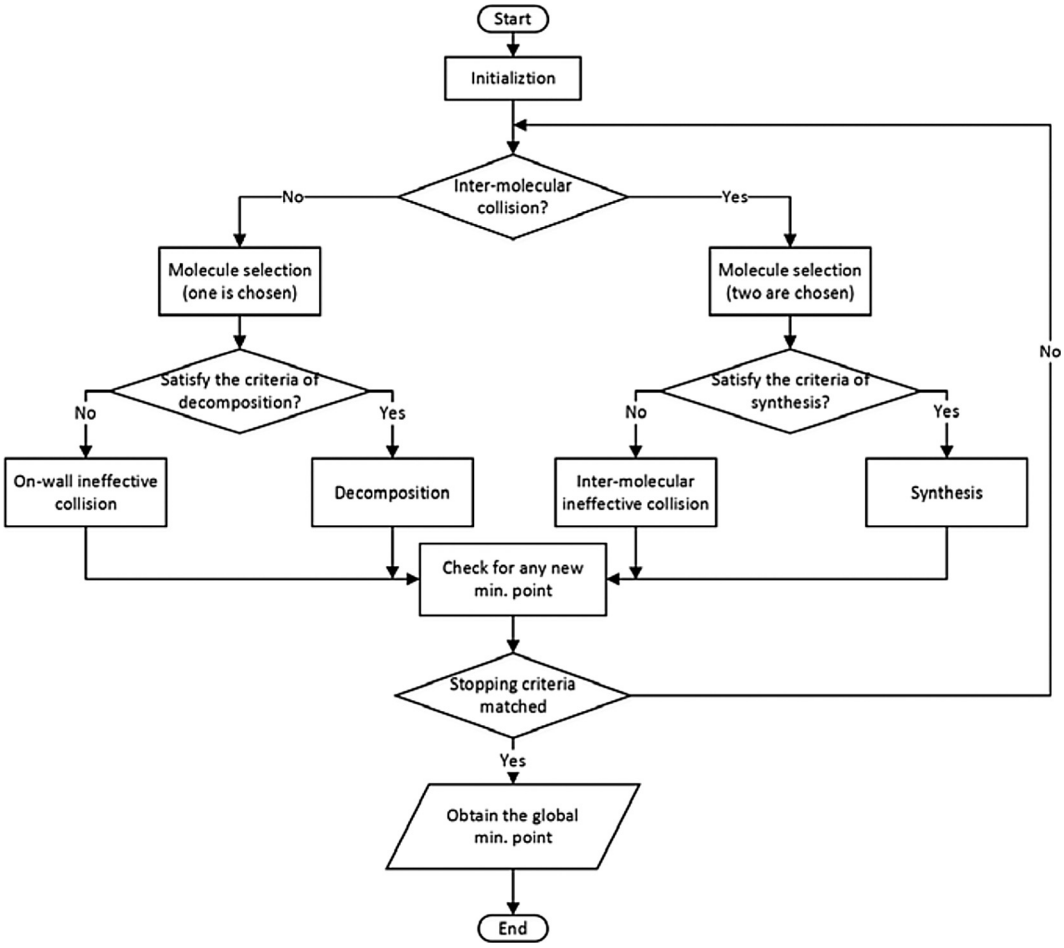

Fig. 2. Flow chart for both the original CRO and enhanced CRO (Lam and Li, 2010).

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
r^h	0	3	2	7	5	1	6	0	1	9	5	0	4	8	0
O^h	10	-3	-7	6	-6	4	-4	0	13	-4	-9	5	3	-8	0

Vehicle 1 Vehicle 2

Fig. 3. Solution representation.

vehicle k, its vehicle type is denoted as $\varepsilon = \overline{k}_k$ (e.g., if the capacities of a fleet of 3 vehicles are $Q_1 = 10$, $Q_2 = 10$, and $Q_3 = 20$, then, $\overline{K} = 2$, $\overline{k}_1 = 1$, $\overline{k}_2 = 1$, $\overline{k}_3 = 2$, $\overline{Q}_1 = 10$, and $\overline{Q}_2 = 20$). The nearby-node set $Nearby_{i,\varepsilon}$ ($i \in V_0$, $\varepsilon \in \{1, ..., \overline{K}\}$) can then be defined as $Nearby_{i,\varepsilon} = \{j \in V_0 \setminus \{i\} | \gamma$ nodes with the smallest $temp_{ij,\varepsilon}\}$, where the value of $temp_{ij,\varepsilon}$ is calculated by

$$temp_{ij,\varepsilon} = \begin{cases} t_{ij}/cP, & j \in V_H \\ t_{ij}/\min(\overline{Q}_\varepsilon, |s_j^0 - q_j|), & j \in \overline{D} \cup V_E \end{cases} \tag{23}$$

The value of γ is determined by Eq. (24). $\lceil T/t' \rceil$ represents the estimated maximum number of nodes that a vehicle can visit within the repositioning duration T, where t' is the average travel time between nodes. Since the service time for loading and unloading operations is counted in the time constraint (refer to constraint (15)), $\lceil T/t' \rceil$ is large enough for estimation. Regarding the multiple-vehicle case, $|K| + 1$ is the factor applied to $\lceil T/t' \rceil$ instead of $|K|$ to allow the inclusion of more nodes in the set in our FFBRP compared to the set in traditional BRPs, because the loading and unloading times required at nodes $i' \in V_H$ are always shorter than those at the nodes $i'' \in \overline{D} \cup V_E$. In addition, the value of γ should not be larger than the total number of nodes excluding the considered node i.

$$\gamma = \min(|V_0|-1, \lceil(|K| + 1)\cdot\lceil T/t'\rceil\rceil), \quad \text{where } t' = \sum_{i,j\in V_0, i\neq j} t_{ij} \Big/ (|V_0|^2-|V_0|) \tag{24}$$

3.3. Initial solution construction

To depict how to obtain the initial solutions, the following notations are defined:

(1) F^h (where $h = SumN + l$, $SumN = \sum_{k'=1}^{k-1}(n_{k'} + 2)$, $l = 0, 1, ..., n_k + 1$, and $k \in K$) represents the number of bikes on vehicle k when it leaves node r^h for r^{h+1}. If $l = 0$, F^h is the number of bikes on vehicle k when it leaves the starting depot. Therefore, if $l = 0$, then $F^h = O^h$. If $l = 1, ..., n_k$, the value of F^h is obtained by $F^h = F^{h-1} + O^h$.
(2) s_i ($i \in V_0$) is the current inventory level at node i after that node is visited by a vehicle and is used to determine the current state of a node. Node i is in a surplus, deficit, and balanced state if s_i-q_i is positive, negative, and zero, respectively.
(3) $TraTime_k$ is the travel time of vehicle k and $OprTime_k$ is the service time for loading and unloading operated by vehicle k.
(4) For all nodes $i \in V_E \cup V_H$, $NearDepot_i = \text{argmin}_{j\in\overline{D}} t_{ij}$ is used to denote the nearest depot from node i.
(5) For each vehicle $k \in K$, $StartDepot_k$ is used to represent the depot where vehicle k must start the repositioning operation. If $a_{ik} = 1$ ($i \in \overline{D}$ and $k \in K$), $StartDepot_k = i$.

An initial solution to the described problem is constructed as follows:

Step 1: Set $s_i = s_i^0$ ($\forall\, i \in V_0$) and $SumN = 0$. Set $k = 1$.
Step 2: Set $r^{SumN} = StartDepot_k$ and $l = 1$. Set $TraTime_k = 0$, $OprTime_k = 0$, and $Temp_k = 0$ ($Temp_k$ is used to identify whether vehicle k can visit more nodes. If $T-(TraTime_k + OprTime_k) < L + U$, set $Temp_k = 1$, which indicates that no more nodes can be visited by vehicle k).
Step 3: If $Temp_k = 0$, nodes are added to the vehicle route and the loading/unloading quantities at added nodes are calculated simultaneously.
Step 4: If $Temp_k = 1$, then set $SumN = SumN + n_k + 2$ and $k = k + 1$.
Step 5: If $k \le |K|$, go to Step 2.
Step 6: If $k > |K|$, stop the procedure.

In Step 3, nodes and their loading/unloading quantities are added as follows:

Step 3.1: Insert a random node $i \in Nearby_{v,\varepsilon}$ (where $v = r^{SumN+l-1}$ and $\varepsilon = \overline{k}_k$) into position $SumN + l$ and set $TraTime_k = TraTime_k + t_{vi}$ only if it satisfies the following:
(1) $|s_i-q_i| \neq 0$;
(2) Time constraint: $tempT = TraTime_k + OprTime_k + t_{vi} + t_{i,NearDepot_i} < T-(L + U)$.
If no node can be inserted, set $l = l - 1$ and $Temp_k = 1$. Go to Step 3.5.
Step 3.2: Calculate the loading/unloading quantities at the inserted node:
Case 1 If $s_i-q_i > 0$ and $i \in V$, the number of bikes loaded onto vehicle k from node i is determined by $O^{SumN+l} = \min(Q_k-F^{SumN+l-1}, s_i-q_i, MaxOpt)$, where $MaxOpt = \lfloor(T-tempT)/(L + U)\rfloor$ is the maximum number of bikes that can be handled within the remaining repositioning time. (To ensure that all the bikes loaded onto the vehicle can be unloaded at the end of the repositioning, the unloading times are reserved when bikes are picked up.) Set $OprTime_k = OprTime_k + (L + U)\cdot O^{SumN+l}$ and $tempT = tempT-(L + U)\cdot O^{SumN+l}$.
Case 2 If $s_i-q_i > 0$ and $i \in \overline{D}$, $O^{SumN+l} = 0$.
Case 3 If $s_i-q_i < 0$, the number of bikes unloaded from the vehicle to node i is $O^{SumN+l} = -\min(F^{SumN+l-1}, q_i-s_i)$.
Step 3.3: Set $s_i = s_i-O^{SumN+l}$ and $F^{SumN+l} = F^{SumN+l-1} + O^{SumN+l}$.
Step 3.4: If $tempT \le L + U$, set $Temp_k = 1$.
Step 3.5: If $Temp_k = 1$, set $l = l + 1$, $r^{SumN+l} = NearDepot_v$, $O^{SumN+l} = -F^{SumN+l-1}$, $TraTime_k = TraTime_k + t_{v,NearDepot_v}$, and $n_k = l-1$. Go to Step 4.
Step 3.6: If $tempT > L + U$, set $l = l + 1$. Go to Step 3.1.

After obtaining the initial solutions, the operators stated in the following Sections 3.3.1 and 3.3.2 are executed to adjust the loading and unloading quantities at the nodes or the depots one by one based on the bike loads along the route.

3.3.1. Adjustment on loading operations

In Step 3 of the initial solution construction procedure, bikes at the visited hardly and easily accessed nodes in a surplus state are loaded onto the vehicle if it has enough space. However, those bikes are only unloaded to the easily accessed nodes which are in the deficit state before the vehicle returns to the ending depot. Therefore, it is possible that the number of bikes loaded onto the vehicle is larger than the number of bikes unloaded. In this case, an excessive number of bikes are loaded and have to be unloaded at the ending depot. *FlowBackRevise* is executed to adjust the loading or unloading quantity at the visited nodes to mitigate the problem of loading an excessive number of bikes. It redistributes the bikes at hardly accessed nodes to easily accessed nodes (either in surplus or deficit)

or reduce the number of bikes loaded at easily accessed nodes.

Before depicting the checking procedure, we define the following notations. $minf_k$ represents the minimum number of bikes on vehicle k when the vehicle travels from the checked node v to the ending depot at the end of repositioning. The depot is denoted as $rtnDepot_k = r^{SumN+n_k+1}$ (where $SumN = \sum_{k'=1}^{k-1}(n_{k'}+2)$). For vehicle $k \in K$, the procedure is depicted as follows:

Step 1: Set $l = n_k$, $v = r^{SumN+l}$, and $tempInd = 0$ (If $tempInd = 1$, it means that there exists an easily accessed node where the bikes loaded at hardly accessed nodes can be unloaded; 0 otherwise.). Set $minf_k = F^{SumN+l}$.

Step 2: If any one of the following conditions is violated, stop the procedure.
(1) $l > 0$; (2) $minf_k > 0$; (3) $O^{SumN+n_k+1} \neq 0$.

Step 3: If $v \in V_E$ and $tempInd = 0$, then set $tempInd = 1$ and $tempP = l$ ($tempP$ is the position where the easily accessed node is saved in the vehicle route).

Step 4: If $v \in V_H$ and $tempInd = 1$, the maximum allowable change in loading/unloading quantities is $\Delta = \min(minf_k, O^{SumN+l}, |O^{SumN+n_k+1}|)$. For node $\bar{v} = r^{SumN+tempP} \in V_E$ and depot $rtnDepot_k$, set (1) $O^{SumN+tempP} = O^{SumN+tempP} - \Delta$ and $O^{SumN+n_k+1} = O^{SumN+n_k+1} + \Delta$; (2) $s_{\bar{v}} = s_{\bar{v}} + \Delta$ and $s_{rntDepot_k} = s_{rntDepot_k} - \Delta$; (3) $F^{SumN+l'} = F^{SumN+l'} - \Delta$, where $l' = tempP, \ldots, n_k$; and (4) $minf_k = minf_k - \Delta$.

Step 5: If $v \in V_E \cup \overline{D}$ and $O^{SumN+l} > 0$, set $\Delta = \min(minf_k, O^{SumN+l}, |O^{SumN+n_k+1}|)$. For node v and depot $rtnDepot_k$, set (1) $O^{SumN+l} = O^{SumN+l} - \Delta$ and $O^{SumN+n_k+1} = O^{SumN+n_k+1} + \Delta$; (2) $s_v = s_v + \Delta$ and $s_{rntDepot_k} = s_{rntDepot_k} - \Delta$; (3) $F^{SumN+l'} = F^{SumN+l'} - \Delta$, where $l' = l, \ldots, n_k$; (4) $minf_k = minf_k - \Delta$; and (5) $OprTime_k = OprTime_k - (L+U)\cdot\Delta$.

Step 6: Set $l = l - 1$ and $minf_k = \min(minf_k, F^{SumN+l})$. Go to Step 2.

In the above procedure, Step 4 attempts to unload the bikes picked up from hardly accessed nodes to easily accessed nodes. Step 5 aims to reduce the excessive number of bikes loaded at depots or easily accessed nodes.

If $O^{SumN+n_k+1} \neq 0$ after implementing the above procedure, some of the loading operations carried out at the hardly accessed nodes need to be removed by the procedure below:

Step 1: Set $l = n_k$ and $v = r^{SumN+l}$.

Step 2: This procedure stops if $O^{SumN+n_k+1} = 0$.

Step 3: If $v \in V_H$ and $O^{SumN+l} > 0$, set $\Delta = \min(O^{SumN+l}, |O^{SumN+n_k+1}|)$. For node v and depot $rtnDepot_k$, set (1) $O^{SumN+l} = O^{SumN+l} - \Delta$ and $O^{SumN+n_k+1} = O^{SumN+n_k+1} + \Delta$; (2) $s_v = s_v + \Delta$ and $s_{rntDepot_k} = s_{rntDepot_k} - \Delta$; (3) $F^{SumN+l'} = F^{SumN+l'} - \Delta$, where $l' = l, \ldots, n_k$; and (4) $OprTime_k = OprTime_k - (L+U)\cdot\Delta$.

Step 4: Set $l = l - 1$, go to Step 2.

3.3.2. Additional calculation for the loading operation at depots

To give priority to handling the bikes at easily and hardly accessed nodes, loading operations at depots are not considered in the construction of initial solutions. Herein, *DepotOperCalculation* is executed to load bikes at depots if there is still space available on the vehicle when it travels to the nodes in deficit. To depict the procedure, we define two more notations: $minrQ_k$ is defined as the minimum residual capacity of vehicle k on the route from the starting depot, denoted as $StartDepot_k$, to the checked nodes. *MaxAdjustT* is the maximum number of bikes that can be handled within the remaining repositioning time (i.e., $T - (TraTime_k + OprTime_k)$). The procedure is stated below.

Step 1: Set $l = 1$, $v = r^{SumN+l}$, $minrQ_k = Q_k - F^{SumN}$, and $MaxAdjustT = \lfloor (T - TraTime_k - OprTime_k)/(L+U) \rfloor$.

Step 2: The procedure continues until any of the following conditions is violated.
(1) $minrQ_k > 0$; (2) $s_{StartDepot_k} > 0$; (3) $MaxAdjustT > 0$; and (4) $l < n_k + 1$.

Step 3: If $v \in V_E$ and $s_v - q_v < 0$, set $\Delta = \min(minrQ_k, |s_v - q_v|, s_{StartDeport_k}, MaxAdjustT)$. For depot $StartDepot_k$ and node v, set (1) $O^{SumN} = O^{SumN} + \Delta$ and $O^{SumN+l} = O^{SumN+l} - \Delta$; (2) $s_{StartDepot_k} = s_{StartDepot_k} - \Delta$ and $s_v = s_v + \Delta$; (3) $F^{SumN+l'} = F^{SumN+l'} + \Delta$, where $l' = 0, \ldots, l-1$; (4) $OprTime_k = OprTime_k + (L+U)\cdot\Delta$; (5) $MaxAdjustT = MaxAdjustT - \Delta$; and (6) $minrQ_k = minrQ_k - \Delta$.

Step 4: Set $l = l + 1$ and $minrQ_k = \min(minrQ_k, Q_k - F^{SumN+l})$. Go to Step 2.

3.4. Solution evaluation

Each solution ω obtained by the enhanced CRO is evaluated by the evaluation function

$$z' = \sum_{i \in V_E} \max(q_i - s_i, 0) + \sum_{i \in V_H} cP \cdot s_i + \mu \cdot \sum_{k \in K} (TraTime_k + OprTime_k) + P \tag{25}$$

where $P = \begin{cases} \sum_{i \in V_E} (\max(q_i - s_i^0, 0)) + \sum_{i \in V_H} cP \cdot s_i^0 + \mu \cdot T, & \text{if } \omega \text{ is infeasible;} \\ 0, & \text{if } \omega \text{ is feasible.} \end{cases}$

The first three terms considered in the evaluation function (25) are the same as those in the objective function (1), including the unmet demand, the total penalty for bikes at hardly accessed nodes, and the weighted vehicles' total operational time. P in (25) is the

penalty associated with an infeasible solution ω due to the *temporary supply*. *Temporary supply* means that the bikes at a node are picked up first regardless of its demand and then its unmet demand is satisfied by other vehicles later. Here is an example of an infeasible solution caused by *temporary supply*. The initial inventory at a node is 6 bikes. Vehicle 1 first picks up 8 bikes at that node and then vehicle 2 unloads 6 bikes there. Although the final inventory level at that node is 4, which is non-negative, the non-negative constraint is violated when bikes are first picked up by vehicle 1.

To determine whether ω is feasible, the following is performed.

Step 1: If there is no node at which a loading operation is carried out by a vehicle while unloading is operated by another vehicle, the solution is feasible and stop.
Step 2: Check whether the solution with temporary supply is feasible.
Step 2.1: Select a node i where both loading and unloading operations occur there.
Step 2.2: Calculate the arrival time $arrTime_k$ at node i by each vehicle $k \in K$. If the node is not visited by vehicle k, set $arrTime_k = T$.
Step 2.3: Determine the firstly arrived vehicle at node i by $k' = \text{argmin}_{k \in K} arrTime_k$.
Step 2.4: The non-negativity inventory constraints (i.e., $s_i \geq 0$) is checked by simply considering the loading/unloading operated by vehicle k' at node i.
Step 2.5: If the constraint is violated, the solution is infeasible and stop checking.
Step 2.6: If all of the nodes with both loading and unloading operations there are checked, stop. Otherwise, go to Step 2.1.

3.5. Four revised elementary reactions

They all incorporate the solution adjustment operators introduced in Section 3.6.

3.5.1. On-wall ineffective collision

One of the vehicle routes in a solution ω is randomly picked and modified when the operator *NewOnwall* is applied. Suppose k is the selected vehicle, $k \in K$. A new solution ω' is obtained without changing the number of nodes saved in the routing sequence. The details of how the operator works are stated below:

Step 1: Identity the position l of the node that leads to the most reduction of evaluation function value if the node is removed from the route by $l = \underset{h=SumN+1,\ldots,SumN+n_k}{\text{argmin}} \{Op + (t_{r^{h-1},r^{h+1}} - (t_{r^{h-1},r^h} + t_{r^h,r^{h+1}} + |O^h| \cdot (L + U))) \cdot \mu\}$,
where $Op = \begin{cases} |O^h|, & r^h \in V_E \\ cP \cdot |O^h|, & r^h \in V_H \end{cases}$, and $SumN = \sum_{k'=1}^{k-1} n_{k'}$.
If more than one node can lead to the same maximal reduction of the evaluation function value, the first node visited by the vehicle is selected and denoted as $v = r^l$.
Step 2: An unbalanced node v' is randomly selected from $Nearby_{i,\varepsilon}$ ($i = r^{l-1}$ and $\varepsilon = \overline{k}_k$) and is used to replace v. The selected node v' must not be the same as r^{l+1} (i.e., $v' \neq r^{l+1}$) to avoid having the same node in the consecutive positions in a route, and must satisfy either one of the following two conditions:
(1) If the loading/unloading quantity at station v is zero, node $v' \neq r^{l+1}$ must satisfy the following: if $s_v - q_v > 0$, then $s_{v'} - q_{v'} < 0$; if $s_v - q_v < 0$, then $s_{v'} - q_{v'} > 0$; if $s_v - q_v = 0$, then $s_{v'} - q_{v'} \neq 0$.
(2) If the loading/unloading quantity at station v is non-zero, node $v' \neq r^{l+1}$ must satisfy the following: if $O^l > 0$, then $s_{v'} - q_{v'} > 0$; if $O^l < 0$, then $s_{v'} - q_{v'} < 0$.
Step 3: Set $TraTime_k = TraTime_k + (t_{r^{l-1},v'} + t_{v',r^{l+1}}) - (t_{r^{l-1},v} + t_{v,r^{l+1}})$.
Step 4: If the time constraint is violated (i.e., $TraTime_k + OprTime_k > T$), *RemovalOfNodesT* (introduced in Section 3.6.3) is applied.
Step 5: *OperationAdj* (Subroutine 5 introduced in Section 3.6.1) is implemented to revise the loading/unloading quantities along the route of vehicle k based on position l.

Fig. 4 shows an example of how *NewOnwall*(ω) works. In this example, vehicle route 1 is modified to get a new solution. Nodes 7, 9, and 10 are hardly accessed nodes and each can provide 1 bike to other nodes. In Step 1, node 10 is identified as the node that leads to the most reduction of evaluation function value and selected to be replaced. Node 8, which is an easily accessed node in the nearby-node set of node 10, is randomly selected in Step 2. Without violating the time constraint, node 10 is replaced by node 8. In Step 5, after *OperationAdj* is executed, both the loading quantities at nodes 7 and 9 are 1 and the unloading quantity at node 10 is 2.

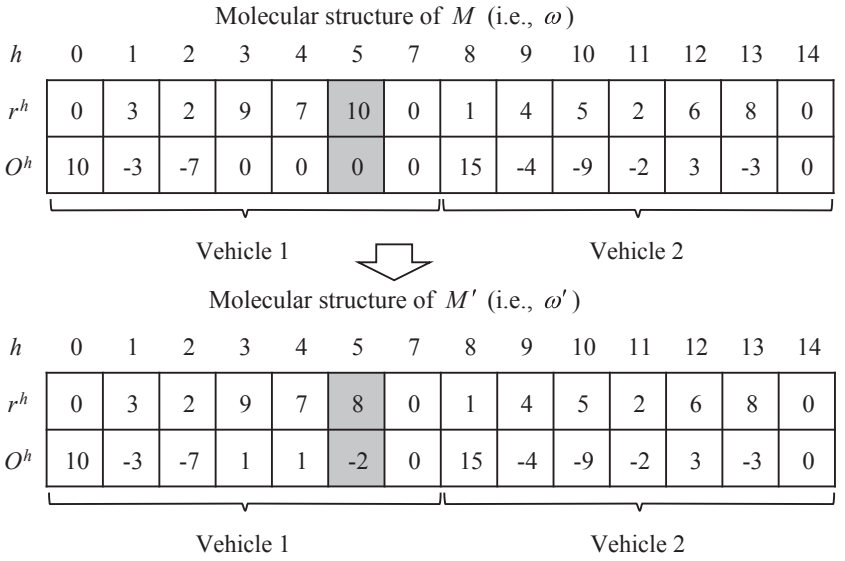

Fig. 4. *NewOnwall*(ω) for the on-wall ineffective collision.

The pseudocode of the revised on-wall ineffective collision is given in Subroutine 1.

Subroutine 1 *NewOnwallIneffCollision*(M, *buffer*)

Input: A molecule M with its profile (i.e., the structure ω, PE_ω, and KE_ω) and the central energy buffer *buffer*

1. Get random $k \in \{1, ..., |K|\}$
2. $\omega' = NewOnwall(\omega)$
3. Calculate $PE_{\omega'}$
4. $(\omega', PE_{\omega'}) = SolutionAdjustment(\omega', PE_{\omega'}, k)$
5. **if** $PE_\omega + KE_\omega \geqslant PE_{\omega'}$ **then**
6. Get q randomly in interval [*KELossRate*, 1]
7. $KE_{\omega'} = (KE_\omega + PE_\omega - PE_{\omega'}) \times q$
8. Update *buffer* = *buffer* + $(KE_\omega + PE_\omega - PE_{\omega'}) \times (1-q)$
9. Update the profile of M by $\omega = \omega'$, $PE_\omega = PE_{\omega'}$, and $KE_\omega = KE_{\omega'}$
10. **end if**
11. **Output** M' and buffer

Remarks: Lines 1 and 4 are not executed in the subroutine *OnwallIneffCollision* in the original CRO, and line 2 in the modified CRO uses the proposed *NewOnwall* operator instead of the original *Onwall* operator introduced by Lam and Li (2010). *KELossRate* refers to the maximum fraction of KE lost. *Buffer* is the central energy buffer which cumulates the energy loss from the on-wall ineffective collision (calculated by the equation in line 8) and is used to ensure the occurrence of decomposition.

3.5.2. Decomposition

Different from *NewOnwall*, *NewDecom* applies to all routes in an original solution ω to obtain two new solutions ω_1' and ω_2'. These new solutions are obtained by the following:

Step 1: Cut each vehicle route into two sub-sequences, $subr_k^1$ and $subr_k^2$ ($\forall\, k \in K$) based on the same position. That position is determined by $\lfloor \min_{k=1,...,|K|} (n_k + 2)/2 \rfloor$.

Step 2: Get new routes r_1' for ω_1': For $k = 1, ..., |K|$, assign sub_k^1 to the corresponding position of vehicle k in r_1'. Then, add nodes selected from the nearby-node set $Nearby_{i,\varepsilon}$ (where i is currently the last node in the route of vehicle k and $\varepsilon = \overline{k}_k$) one by one starting from the first position after the cutting point until the number of nodes in the route reaches $\min_{k=1,...,|K|} n_k$. Based on the last visited node, add its nearest depot to the end of the route.

Step 3: Get new routes r_2' for ω_2': For $k = 1, ..., |K|$, assign $StartDepot_k$ to the beginning of each route, and assign sub_k^2 to the corresponding position of vehicle k in r_2' except for the ending depot. Then, the nodes selected from the nearby-node set $Nearby_{i,\varepsilon}$ (where i is currently the last node in the route of vehicle k and $\varepsilon = \overline{k}_k$) are added to the route one by one until the number of nodes reaches $\min_{k=1,...,|K|} n_k$. Based on the last visited node, add its nearest depot to the end of the route.

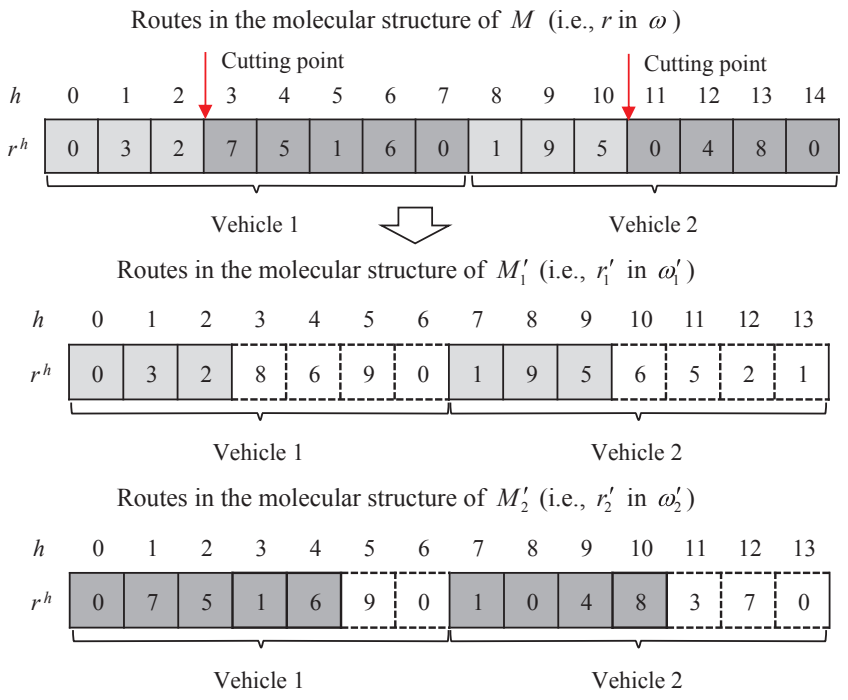

Fig. 5. *NewDecom*(ω) for decomposition.

Step 4: For each new route, calculate the loading/unloading quantity at each node and update $OprTime_k$ by *OperationDetermination* (Subroutine 8 introduced in Section 3.7).

Fig. 5 shows how the operator *NewDecom*(ω) works on the vehicle routes (i.e., Steps 1–3). In the original solution, $\lfloor \min_{k=1,\ldots,|K|} (n_k + 2)/2 \rfloor = 3$. Therefore, in Fig. 5, the sub-sequences in light grey are the sub-sequences sub_1^1 and sub_2^1, whereas the sub-sequences in grey are sub_1^2 and sub_2^2. The lengths of the two subsequences sub_1^1 and sub_2^1 are both equal to 3. Moreover, $\min_{k=1,\ldots,|K|} n_k = 5$. Hence, the length of each route in the two new solutions is $\min_{k=1,\ldots,|K|} n_k + 2 = 7$. For the first new solution, 7 – 3 more nodes are added to both vehicle routes 1 and 2 separately, and for the second new solution, 7 – 5 more nodes are added to both vehicle routes 1 and 2 separately. The nodes in the dotted boxes in Fig. 5 are the newly added. After decomposition, vehicle route 1 is shorter and has the same length as vehicle route 2. This operator allows making a route shorter, giving more time for loading and unloading at the other visited nodes.

The pseudocode of the decomposition in the enhanced CRO is presented in Subroutine 2.

Subroutine 2 *NewDecomposition*(*M*, *buffer*)

Input: A molecule M with its profile (i.e., the structure ω, PE_ω, and KE_ω) and the central energy buffer *buffer*

1. $(\omega_1', \omega_2') = NewDecom(\omega)$
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. **for** each vehicle $k \in \{1, \ldots, |K|\}$ **do**
4. $(\omega_1', PE_{\omega_1'}) = SolutionAdjustment(\omega_1', PE_{\omega_1'}, k)$
5. $(\omega_2', PE_{\omega_2'}) = SolutionAdjustment(\omega_2', PE_{\omega_2'}, k)$
6. **end for**
7. Let $temp_1 = PE_\omega + KE_\omega - PE_{\omega_1'} - PE_{\omega_2'}$
8. Create a Boolean variable *Success*
9. **if** $temp_1 \geqslant 0$ **then**
10. *Success* = TRUE
11. Create two new molecules M_1' and M_2'
12. Get $\overline{m}$ randomly in interval [0, 1]
13. $KE_{\omega_1'} = temp_1 \times \overline{m}$
14. $KE_{\omega_2'} = temp_1 \times (1 - \overline{m})$

15. Assign ω_1', $PE_{\omega_1'}$, and $KE_{\omega_1'}$ to the profile of M_1', and ω_2', $PE_{\omega_2'}$, and $KE_{\omega_2'}$ to the profile of M_2'
16. **else if** $temp_1 + buffer \geqslant 0$ **then**
17. $Success$ = TRUE
18. Get m_1 m_2, m_3, and m_4 independently randomly in interval [0,1]
19. $KE_{\omega_1'} = (temp_1 + buffer) \times m_1 \times m_2$
20. $KE_{\omega_2'} = (temp_1 + buffer) \times m_3 \times m_4$
21. Update $buffer = temp_1 + buffer - KE_{\omega_1'} - KE_{\omega_2'}$
22. Assign ω_1', $PE_{\omega_1'}$, and $KE_{\omega_1'}$ to the profile of M_1', and ω_2', $PE_{\omega_2'}$, and $KE_{\omega_2'}$ to the profile of M_2'
23. **else**
24. $Success$ = FALSE
25. **end if**
26. **Output** M_1', M_2', $Success$, and $buffer$

Remarks: Lines 3 through 6 are not executed in the subroutine *Decomposition* in the original CRO, and line 1 in the modified CRO uses the proposed *NewDecom* operator instead of the original *Decom* operator. The original *Decom* operator uses the circular shift operator introduced by Lam and Li (2010) to obtain new solutions.

3.5.3. Intermolecular ineffective collision

The *NewInter* operator guides the procedure on how to get two new solutions from two original solutions. For each original solution, only one vehicle route is modified. Let it be the route of vehicle k. From each original solution, a new solution is obtained by swapping two sub-sequences ($subr_1$ and $subr_2$) with equal length. The method works as follows:

Step 1: Determine the maximum length of the sub-sequences for swapping by $\lfloor n_k/2 \rfloor - 1$.
Step 2: Determine the common length of sub-sequences randomly, $length \in \{1, ..., \lfloor n_k/2 \rfloor - 1\}$.
Step 3: If $length = 1$, two non-adjacent nodes are selected from the route and swapped. Their positions are l_1 and l_2, where $l_1, l_2 \in \{1, ..., n_k\}$.
Step 4: If $length > 1$, two sub-sequences are randomly selected and swapped. Then,
(i) Randomly select the starting position l_2 of $subr_2$ by $l_2 \in \{1 + length, ..., n_k - 1 - length\}$.
(ii) Randomly select the starting position l_1 of $subr_1$ by $l_1 \in \{1, ..., l_2 - length\}$.
Step 5: Update the travel time.
The first nodes in the two sub-sequences $subr_1$ and $subr_2$ are denoted as $v_1 = r^{SumN+l_1}$ and $v_2 = r^{SumN+l_2}$, respectively. Their last nodes are defined as $\bar{v}_1 = r^{SumN+l_1+length-1}$ and $\bar{v}_2 = r^{SumN+l_2+length-1}$, respectively. If $l_1 + length \neq l_2$, perform the steps in case 1, and perform the steps in case 2 otherwise.
Case 1: $l_1 + length \neq l_2$
(i) If $length = 1$, set $\bar{v}_1 = v_1$ and $\bar{v}_2 = v_2$.
(ii) The preceding nodes of v_1 and v_2 are respectively $pred(v_1) = r^{SumN+l_1-1}$ and $pred(v_2) = r^{SumN+l_2-1}$.
(iii) The succeeding nodes of $\bar{v}_1$ and $\bar{v}_2$ are respectively $sucd(\bar{v}_1) = r^{SumN+l_1+length}$ and $sucd(\bar{v}_2) = r^{SumN+l_2+length}$.
The travel time of vehicle k is updated by $TraTime_k = TraTime_k - tDif$, where $tDif = t_{pred(v_1),v_2} + t_{\bar{v}_2,sucd(\bar{v}_1)} + t_{pred(v_2),v_1} + t_{\bar{v}_1,sucd(\bar{v}_2)} - (t_{pred(v_1),v_1} + t_{\bar{v}_1,sucd(\bar{v}_1)} + t_{pred(v_2),v_2} + t_{\bar{v}_2,sucd(\bar{v}_2)})$.
Case 2: $l_1 + length = l_2$
(i) Set $sucd(\bar{v}_1) = v_2$ and $pred(v_2) = \bar{v}_1$.
(ii) The travel time of vehicle k is updated by $TraTime_k = TraTime_k - tDif$, where $tDif = (t_{pred(v_1),v_2} + t_{\bar{v}_2,v_1} + t_{\bar{v}_1,sucd(\bar{v}_2)}) - (t_{pred(v_1),v_1} + t_{\bar{v}_1,v_2} + t_{\bar{v}_2,sucd(\bar{v}_2)})$.

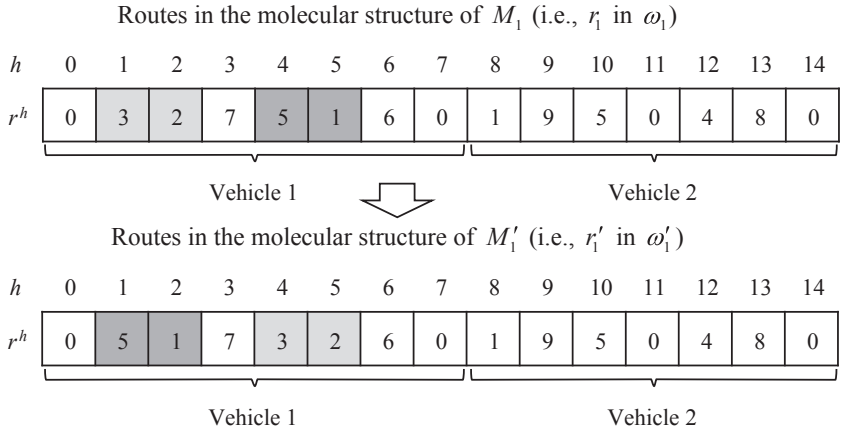

Fig. 6. *NewInter*(ω_1, ω_2) for the inter-molecular ineffective collision.

Step 6: If $T-(TraTime_k + OprTime_k) > L + U$, calculate the loading/unloading quantity at each node and update the total service time $OprTime_k$ by *OperationDetermination* (Subroutine 8 introduced in Section 3.7); otherwise, no feasible solution can be obtained from *NewInter* and the reaction will not occur.

Fig. 6 gives an example for *NewInter*. In this example, vehicle route 1 is selected and modified. The lengths of the swapped subsequences are both equal to 2.

The pseudocode of the intermolecular ineffective collision is given in Subroutine 3.

Subroutine 3 *NewInterMoleIneffCollision*(M_1, M_2)

Input: Molecule M_1 with its profile and M_2 with its profile

1. Get random $k_1, k_2 \in \{1, ..., |K|\}$
2. $(\omega_1', \omega_2') = NewInter(\omega_1, \omega_2)$
3. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
4. $(\omega_1', PE_{\omega_1'}) = SolutionAdjustment(\omega_1', PE_{\omega_1'}, k_1)$
5. $(\omega_2', PE_{\omega_2'}) = SolutionAdjustment(\omega_2', PE_{\omega_2'}, k_2)$
6. Let $temp_2 = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega_1'} - PE_{\omega_2'}$
7. **if** $temp_2 \geqslant 0$ **then**
8. Get $\overline{m}$ randomly in interval $[0, 1]$
9. $KE_{\omega_1'} = temp_2 \times \overline{m}$
10. $KE_{\omega_2'} = temp_2 \times (1-\overline{m})$
11. Update the profile of M_1 by $\omega_1 = \omega_1'$, $PE_{\omega_1} = PE_{\omega_1'}$, and $KE_{\omega_1} = KE_{\omega_1'}$, and the profile of M_2 by $\omega_2 = \omega_2'$, $PE_{\omega_2} = PE_{\omega_2'}$, and $KE_{\omega_2} = KE_{\omega_2'}$
12. **end if**
13. **Output** M_1', M_2'

Remarks: Lines 1, 4 – 5 are not executed in the subroutine *InterMoleIneffCollision* in the original CRO, and line 2 in the modified CRO uses the proposed *NewInter* operator instead of the original *Inter* operator introduced by Lam and Li (2010).

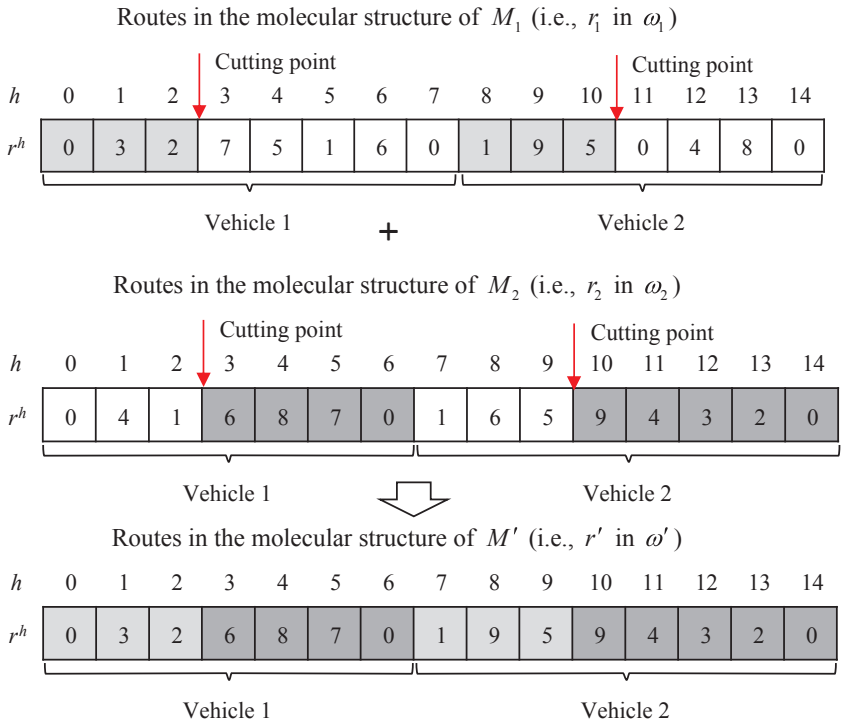

Fig. 7. *NewSynth*(ω_1, ω_2) for the synthesis.

3.5.4. Synthesis

NewSynth(ω_1, ω_2) is used to generate a new solution by combining the routes in two existing solutions ω_1 and ω_2, and then determine the corresponding loading and unloading quantities at each visited node. The method to obtain the new solution is described as follows:

Step 1: For solutions ω_1 and ω_2, each route is cut into two sub-sequences, denoted as $\omega 1subr_k^1$ and $\omega 1subr_k^2$ for ω_1, and $\omega 2subr_k^1$ and $\omega 2subr_k^2$ for ω_2 ($k = 1, ..., |K|$) based on the same position. That position is determined by $\left\lfloor \min_{k=1,...,|K|} (n_k + 2)/2 \right\rfloor$.

Step 2: As illustrated in Fig. 7, for each vehicle $k = 1, ..., |K|$, combine $\omega 1subr_k^1$ and $\omega 2subr_k^2$ to get new routes and calculate $TraTime_k$.

Step 3: For each new route, if $T - TraTime_k < 0$, *RemovalOfNodesT* (introduced in Section 3.6.3) is executed to remove node(s).

Step 4: For each new route, *OperationDetermination* (Subroutine 8 introduced in Section 3.7) is applied to determine the loading and unloading quantities and the operational time of the vehicle.

The pseudocode of synthesis is presented in Subroutine 4.

Subroutine 4 *NewSynthesis*(M_1, M_2)

Input: Molecule M_1 with its profile and M_2 with its profile

1. $\omega' = NewSynth(\omega_1, \omega_2)$
2. Calculate $PE_{\omega'}$
3. **for** each vehicle k **do**
4. $(\omega', PE_{\omega'}) = SolutionAdjustment(\omega', PE_{\omega'}, k)$
5. **end for**
6. Create a Boolean variable *Success*
7. **if** $PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geqslant PE_{\omega'}$ **then**
8. *Success* = TRUE
9. Create one molecule M'
10. $KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}$
11. Assign ω', $PE_{\omega'}$, and $KE_{\omega'}$ to the profile of M'
12. **else**
13. *Success* = FALSE
14. **end if**
15. **Output** M' and *Success*

Remarks: Lines 3 – 5 are not executed in the subroutine *Synthesis* in the original CRO, and line 1 in the modified CRO uses the proposed *NewSynth* operator instead of the original *Synth* operator introduced by Lam and Li (2010).

3.6. Solution adjustments

In this section, five operators are introduced to adjust routes or loading/unloading quantities in the elementary reactions mentioned in Section 3.5. Among these five operators, the operator introduced in Section 3.6.1 is used to improve loading/unloading quantities and the other four operators introduced in Sections 3.6.2–3.6.5 are used to adjust routes. The main ideas of route adjustment include (1) adjusting route lengths (i.e., the number of nodes in routes), including insertions and removals of nodes (2) repairing infeasible solutions (due to the violation of the operational time constraint) by deleting nodes, and (3) rearranging the order of the visited nodes by 2-*Opt* to improve the solution quality.

3.6.1. Adjustments to loading and unloading quantities

The quality of a solution ω may be improved by adjusting the loading/unloading quantities assigned to the nodes in r. The applied adjustments differ based on the type of node. Subroutine 5 shows the adjustment framework. If the node concerned is an easily accessed and imbalanced node, *OperationAdjEasy* is first implemented, followed by *OperationAdjHard* (see Subroutine 5, line 1 – 4). *OperationAdjHard* is also used because a large penalty is assigned to each bike parked at hardly accessed nodes and improvements in the loading/unloading quantities at visited hardly accessed nodes lead to a reduction in the total penalty assigned to bikes at hardly accessed nodes in the objective function. If the node is a hardly accessed node, its loading or unloading quantity will only be checked by *OperationAdjHard* (see Subroutine 5, line 5). The details of *OperationAdjEasy* and *OperationAdjHard* are respectively depicted in Sections 3.6.1.1 and 3.6.1.2, where they involve the following notations defined below:

(1) $adjust_v$ represents the maximum allowable increment (in terms of bikes) in the loading/unloading quantity at the checked node v.
(2) $SumAdjust_v$ is the cumulative increment in the loading/unloading quantity at node v.
(3) $MaxAdjust_v$ limits the maximum increment in the loading/unloading quantity at node v to the number of bikes at which node v

can provide or receive within the remaining time in order to increase demand satisfaction, i.e., $\min(|s_v - q_v|, MaxAdjustT)$. By definition, $SumAdjust_v \leqslant MaxAdjust_v$

(4) $minrQ_k$ is defined as the minimum residual capacity of vehicle k on the route segment considered.

(5) $minf_k$ is the minimum number of the bikes on vehicle k on the route segment considered.

Subroutine 5 *OperationAdj*()

Input: A molecular structure ω, vehicle k, and the position of considered node l

1. **if** $v = r^{SumN+l} \in V_E$ and $s_v - q_v \neq 0$ **then**
2. $\omega' = OperationAdjEasy(\omega, k, l)$
3. $\omega' = OperationAdjHard(\omega', k)$
4. **else if** $v = r^{SumN+l} \in V_H$
5. $\omega' = OperationAdjHard(\omega, k)$
6. **end if**
7. **Output** ω'

3.6.1.1. Loading and unloading adjustments invoked by an easily accessed node. Suppose node $v \in V_E$ is the l-th node visited by vehicle k (i.e., $v = r^{SumN+l}$, where $SumN = \sum_{k'=1}^{k-1}(n_{k'} + 2)$). The inventory level at that node after the repositioning s_v is calculated and used to determine the current condition of each node. If node v is balanced (i.e., $s_v - q_v = 0$), no adjustment on loading/unloading is necessary; otherwise, the operator *OperationAdjEasy* is applied. For an imbalanced node v, either one of the following two states (i.e., surplus and deficit states) depicted in Sections 3.6.1.1.1 and 3.6.1.1.2 will occur.

3.6.1.1.1. At surplus node v. If node $v = r^{SumN+l}$ (i.e., $h = SumN + l$) is currently in a surplus state (i.e., $s_v - q_v > 0$), node v can provide some additional bikes to its succeeding stations (i.e., $sucd(v) = r^{SumN+l'}, l' = l + 1, \ldots, n_k$) in a deficit state to reduce the unmet demand. The adjustments of loading and unloading at node v and the succeeding stations depends on $MaxAdjust_v = \min(|s_v - q_v|, MaxAdjustT)$, where $MaxAdjustT$ is determined by Eq. (26) stated below.

$$MaxAdjustT = \begin{cases} \lfloor (T - TraTime_k - OprTime_k)/(L + U) \rfloor, & \text{if } O^h \geqslant 0 \\ \lfloor (T - TraTime_k - OprTime_k)/(L + U) \rfloor + |O^h|, & \text{if } O^h < 0 \end{cases} \tag{26}$$

For the case $O^h \geqslant 0$ at a currently surplus node, $MaxAdjustT$ is simply calculated by the remaining repositioning time divided by the time required by loading and unloading a bicycle. For the case $O^h < 0$, unloading at a surplus node means that extra bikes are unloaded at node v. $|O^h|$ fewer bikes should be unloaded at node v and $|O^h|$ more bikes should then be transported to other deficit nodes. Therefore, $|O^h|$ is added back to $MaxAdjustT$ when $O^h < 0$.

If there is still space available on vehicle k after the vehicle passes node v, node v can provide some additional bikes to its succeeding stations in a deficit state. The succeeding nodes are checked one by one and their unloading quantities are adjusted based on the following steps:

Step 1: Set $h' = h + 1$, $SumAdjust_v = 0$, $sucd(v) = r^{h'}$, and $minrQ_k = Q_k - F^h$.

Step 2: If any one of the following conditions is satisfied, stop checking the succeeding nodes of node v and move to check its preceding nodes.
(1) $minrQ_k = 0$;
(2) $h' \geqslant n_k + 1$;
(3) $SumAdjust_v = MaxAdjust_v$.

Step 3: If $s_{sucd(v)} - q_{sucd(v)} < 0$, do the following:
Step 3.1: Set $adjust_v = \min(minrQ_k, MaxAdjust_v - SumAdjust_v, |s_{sucd(v)} - q_{sucd(v)}|)$.
Step 3.2: Update the total loading and unloading time by the following equations:

$$OprTime_k = OprTime_k + \begin{cases} (L + U) \cdot adjust_v, & \text{if } O^h \geqslant 0 \\ (L + U) \cdot \max(O^h + adjust_v, 0), & \text{if } O^h < 0 \end{cases} \tag{27}$$

$$OprTime_k = OprTime_k + \begin{cases} -(L + U) \cdot \min(O^{h'}, adjust_v), & \text{if } O^{h'} > 0 \\ 0, & \text{if } O^{h'} \leqslant 0 \end{cases} \tag{28}$$

Step 3.3: Set $s_v = s_v - adjust_v$, $s_{sucd(v)} = s_{sucd(v)} + adjust_v$, $O^h = O^h + adjust_v$, $O^{h'} = O^{h'} - adjust_v$, $SumAdjust_v = SumAdjust_v + adjust_v$, $minrQ_k = minrQ_k - adjust_v$, $F^g = F^g + adjust_v$, where $g = h, \ldots, h' - 1$.

Step 4: Set $minrQ_k = \min(minrQ_k, Q_k - F^{h'})$ and $h' = h' + 1$. Go to Step 2.

Note that according to Eq. (27), if $O^h \geqslant 0$, the time needed for loading and unloading extra $adjust_v$ bikes is directly added to $OprTime_k$. If $O^h < 0$, two cases are considered: (i) $O^h + adjust_v \leqslant 0$ means that the unloading operation is still carried out at node v after the adjustment. Some of the unloaded bikes at node v are shifted to the checked deficit node $sucd(v)$ on the route, and the total number of bikes loaded onto vehicle k remains unchanged. Therefore, $OprTime_k$ does not change. (ii) $O^h + adjust_v > 0$ means that the operation at node v turns out to be a loading instead of unloading one. It implies that the total number of bikes loaded along the route increases and so is $OprTime_k$. From Eq. (28), if loading originally occurs at the checked deficit node $sucd(v)$, the deficit condition of $sucd(v)$ is at least partially caused by loading an excessive number of bikes. Therefore, after the adjustment, these extra bikes should not be loaded at $sucd(v)$ and the handling time for these bikes should be deducted and is modeled by the minus term in Eq. (28). If unloading originally occurs at node $sucd(v)$, more bikes are unloaded after the adjustment. The time required for the additional unloading operation is already counted in Eq. (27), and therefore there is no change in $OprTime_k$ for the case $O^{h'} \leqslant 0$.

If there are some bikes on vehicle k before it arrives at node v and $SumAdjust_v < MaxAdjust_v$, some of the bikes on the vehicle should be first unloaded to supply bikes to the preceding nodes in a deficit state and the same number of bikes should be loaded from node v later. The preceding nodes and their loading quantities are determined by the following steps:

Step 1: Set $h' = h-1$. The node at position h' is $pred(v) = r^{h'}$. Set $minf_k = F^{h'}$.
Step 2: If any one of the following conditions is satisfied, stop.
(1) $minf_k = 0$;
(2) $h' \leqslant 0$;
(3) $SumAdjust_v = MaxAdjust$.
Step 3: If $s_{pred(v)}-q_{pred(v)} < 0$, do the following
Step 3.1: Set $adjust_v = \min(minf_k, MaxAdjust_v-SumAdjust_v, |s_{pred(v)}-q_{pred(v)}|)$.
Step 3.2: Update $OprTime_k$ by Eqs. (27) and (28).
Step 3.3: Set $s_v = s_v-adjust_v$, $s_{pred(v)} = s_{pred(v)} + adjust_v$, $O^h = O^h + adjust_v$, $O^{h'} = O^{h'}-adjust_v$, $SumAdjust_v = SumAdjust_v + adjust_v$, $minf_k = minf_k-adjust_v$, and $F^g = F^g-adjust_v$, where $g = h', \ldots, h-1$.
Step 4: Set $minf_k = \min(minf_k, F^{h'-1})$, $h' = h'-1$. Go to Step 2.

3.6.1.1.2. At deficit node v. If node $v = r^{SumN+l}$ is currently in a deficit state (i.e., $s_v-q_v < 0$), the adjustment on the loading quantities at the succeeding and preceding stations in a surplus condition can contribute to reducing the unmet demand at node v. Like Section 3.6.1.1.1, $h = SumN + l$. Unlike that subsection, $MaxAdjustT$ is determined by Eq. (29) stated below.

$$MaxAdjustT = \begin{cases} \lfloor (T-TraTime_k-OprTime_k)/(L+U) \rfloor + O^h, & \text{if } O^h \geqslant 0 \\ \lfloor (T-TraTime_k-OprTime_k)/(L+U) \rfloor, & \text{if } O^h < 0 \end{cases} \tag{29}$$

For the case $O^h \geqslant 0$, the deficit condition at node v is caused by loading an excessive number of bikes. To reduce the unmet demand at node v, O^h fewer bikes are loaded and therefore O^h is added to $MaxAdjustT$. For $O^h < 0$, no adjustment on $MaxAdjustT$ is needed.

If vehicle k carries bikes after it leaves node v, some bikes should first be unloaded at node v and the same number of bikes should be loaded onto the vehicle from the *succeeding nodes* of node v later. The detailed adjustment procedure for this case is stated as follows:

Step 1: Set $h' = h + 1$, $sucd(v) = r^{h'}$, $minf_k = F^h$, and $SumAdjust_v = 0$.
Step 2: If any one of the following conditions is satisfied, stop.
(1) $minf_k = 0$;
(2) $h' \geqslant n_k + 1$;
(3) $SumAdjust_v = MaxAdjust_v$.
Step 3: If $s_{sucd(v)}-q_{sucd(v)} > 0$, do the following:
Step 3.1: Set $adjust_v = \min(minf_k, MaxAdjust_v-SumAdjust_v, s_{sucd(v)}-q_{sucd(v)})$.
Step 3.2: Update $OprTime_k$ by the following equations:

$$OprTime_k = OprTime_k + \begin{cases} (L+U)\cdot\max(adjust_v-O^h, 0), & \text{if } O^h \geqslant 0 \\ (L+U)\cdot adjust_v, & \text{if } O^h < 0 \end{cases} \tag{30}$$

$$OprTime_k = OprTime_k + \begin{cases} 0, & \text{if } O^{h'} \geqslant 0 \\ -(L+U)\cdot\min(|O^{h'}|, adjust_v), & \text{if } O^{h'} < 0 \end{cases} \tag{31}$$

Step 3.3: Set $s_v = s_v + adjust_v$, $s_{sucd(v)} = s_{sucd(v)}-adjust_v$, $O^h = O^h-adjust_v$, $O^{h'} = O^{h'} + adjust_v$, $minf_k = minf_k-adjust_v$, $SumAdjust_v = SumAdjust_v + adjust_v$, and $F^g = F^g-adjust_v$, where $g = h, \ldots, h'-1$.
Step 4: Set $minf_k = \min(minf_k, F^{h'})$ and $h' = h' + 1$. Go to Step 2.

Regarding Eq. (30), if $O^h \geqslant 0$ and $O^h - adjust_v \geqslant 0$, loading bikes occurs at node v before and after the adjustment. After the adjustment, $adjust_v$ bikes originally loaded at node v are loaded at node $sucd(v)$, and thus the total number of bikes loaded onto vehicle k remains the same, so does the $OprTime_k$. If $O^h \geqslant 0$ and $O^h - Adjust_v < 0$, the loading operation at node v turns out to be an unloading operation at node v after the adjustment. This change implies that additional $adjust_v - O^h$ bikes are loaded from node $sucd(v)$ in a surplus state and $OprTime_k$ increases by $(adjust_v - O^h)\cdot(L + U)$. If $O^h < 0$, the time needed for loading and unloading extra $adjust_v$ bikes is directly added to $OprTime_k$ since more bikes are handled by the vehicle after the adjustment. From Eq. (31), there is no change in $OprTime_k$ if loading is operated at the checked surplus node $sucd(v)$. This is because more bikes are loaded after the adjustment and the time required for loading is already counted in Eq. (30). If unloading occurs at the checked surplus node, the surplus condition is at least partially caused by unloading an excessive number of bikes. Thus, after the adjustment, a decrease in the unloading quantity at node $sucd(v)$ leads to a reduction in the loading and unloading time.

If there are some spaces available on vehicle k for loading more bikes on its way to node v and $SumAdjust_v < MaxAdjust_v$, more bikes should be loaded onto that vehicle from the preceding nodes in a surplus state. The detailed adjustment procedure is stated as follows:

Step 1: Set $h' = h-1$. The node at position h' is $pred(v) = r^{h'}$. Set $minrQ_k = Q_k - F^{h'}$.
Step 2: If any one of the following conditions is satisfied, stop.
(1) $minrQ_k \leqslant 0$;
(2) $h' \leqslant -1$;
(3) $SumAdjust_v = MaxAdjust_v$.
Step 3: If $s_{pred(v)} - q_{pred(v)} > 0$, do the following:
Step 3.1: Set $adjust_v = \min(minrQ_k, MaxAdjust_v - SumAdjust_v, s_{pred(v)} - q_{pred(v)})$.
Step 3.2: Update the loading and unloading time by Eqs. (30) and (31).
Step 3.3: Set $s_v = s_v + adjust_v$, $s_{pred(v)} = s_{pred(v)} - adjust_v$, $O^h = O^h - adjust_v$, $O^{h'} = O^{h'} + adjust_v$, $SumAdjust_v = SumAdjust_v + adjust$, $minrQ_k = minrQ_k - adjust_v$, and $F^g = F^g + adjust_v$, where $m = h', \ldots, h-1$.
Step 4: Set $minrQ_k = \min(minrQ_k, Q_k - F^{h'-1})$, $h' = h'-1$. Go to Step 2.

3.6.1.2. Loading and unloading adjustments invoked by a hardly accessed node. Let R be the set that contains all the nodes visited by vehicle k. Define the ordered sets $H^0 = \{i \in V_H \cap R | s_i = 0\}$, $H^+ = \{i \in V_H \cap R | s_i > 0\}$, and $H^- = \{i \in V_H \cap R | s_i < 0\}$. The nodes are saved in the ascending order based on their position l in the route of vehicle k. Hl^0, Hl^+, and Hl^- store the corresponding position l of the nodes in H^0, H^+, and H^-, respectively. For example, $H_g^+ = r^{SumN + Hl_g^+}$, where $SumN = \sum_{k'=1}^{k-1}(n_{k'} + 2)$ and $g \in \{1, \ldots, |H^+|\}$.

The adjustment procedure in this section considers three cases: $H^0 \neq \emptyset$, $H^+ \neq \emptyset$ and $H^- \neq \emptyset$. $H^0 \neq \emptyset$ means that there exist some hardly accessed nodes with non-zero inventories after repositioning. In this case, there is no need to check those nodes, since no penalties are induced from those nodes. $H^+ \neq \emptyset$ indicates that some bikes are stored at the hardly accessed nodes visited by vehicle k and it may be possible to redistribute these bikes to easily accessed nodes. $H^- \neq \emptyset$ implies that there exist some nodes where the number of bikes loaded at each of these nodes is larger than the number of bikes provided there. This infeasible solution may be caused by the implementation of the reaction operator (introduced in Section 3.5.1). Thus, it may be fixed by loading more bikes from other nodes visited by vehicle k. The loading/unloading quantities at nodes in H^+ and H^- are all checked in this procedure. This is because if the loading/unloading quantities at the nodes in H^+ and H^- can be improved, the objective function value is reduced greatly.

The adjustment details for the cases $H^+ \neq \emptyset$ and $H^- \neq \emptyset$ are depicted in Sections 3.6.1.2.1 and 3.6.1.2.2, respectively, where the initial value for *MaxAdjustT* is determined directly by the remaining repositioning time.

3.6.1.2.1. For the case $H^+ \neq \emptyset$. In each step of the procedure of this case, all the nodes in H^+ are checked one by one. The checking procedure is stated as follows.

Step 1: Check from the considered node $v \in H^+$ to the end of the route: If $minrQ_k > 0$ and $MaxAdjustT > 0$, it is possible to load more bikes onto the vehicle from node v and transport them to its succeeding node(s) in a deficit state to reduce the unmet demand and total penalty assigned to bikes parked at $v \in V_H$.
Step 2: If $H^+ \neq \emptyset$, check from the considered node $v \in H^+$ to the beginning of the route. If $minf_k > 0$, it is possible that fewer bikes at the preceding node(s) of v are loaded and the bikes at v are loaded instead to reduce the total penalty assigned to bikes parked at $v \in V_H$. In this step, the unmet demand is not reduced.
Step 3: If $H^+ \neq \emptyset$, check from the considered node $v \in H^+$ to the end of the route. If the succeeding node of v is a depot, it is possible that the bikes at node v are loaded first and fewer bikes are loaded at the depot without an increment in $OprTime_k$. If the succeeding node of v is an easily accessed node, it is possible that the bikes at node v are redistributed to the easily accessed node with $OprTime_k$ increased to reduce the total penalty assigned to bikes parked at $v \in V_H$, but not the unmet demand.

In Step 1, the succeeding nodes of $v \in H^+$ are checked as follows:

Step 1.1: Set $g = 1$ and $Temph' = 0$. ($Temph'$ represents the last position of all the nodes in the route which has been checked, and it is used to avoid repeated checking.)
Step 1.2: Set $v = H_g^+$, $l = Hl_g^+$, $h = SumN + l$, $h' = \max(SumN + l + 1, Temph')$, $sucd(v) = r^{h'}$, and $minrQ_k = Q_k - F^h$.

Step 1.3: If $minrQ_k = 0$, $g = g + 1$ and go to Step 1.2.
Step 1.4: If either of the following conditions is satisfied, go to Step 2.
(1) $h' \geqslant n_k + 1$;
(2) $MaxAdjustT = 0$.
Step 1.5: If $s_{sucd(v)} - q_{sucd(v)} < 0$, set $adjust_v = \min(s_v, |s_{sucd(v)} - q_{sucd(v)}|, minrQ_k, MaxAdjustT)$.
Step 1.6: Update the total loading and unloading time and $MaxAdjustT$ by the following:

$$OprTime_k = OprTime_k + \begin{cases} (L + U)\cdot\max(adjust_v - O^{h'}, 0), & \text{if } O^{h'} > 0 \\ (L + U)\cdot adjust_v, & \text{if } O^{h'} \leqslant 0 \end{cases} \tag{32}$$

$$MaxAdjustT = MaxAdjustT - \begin{cases} \max(adjust_v - O^{h'}, 0), & \text{if } O^{h'} > 0 \\ adjust_v, & \text{if } O^{h'} \leqslant 0 \end{cases} \tag{33}$$

Set $s_v = s_v - adjust_v$, $s_{sucd(v)} = s_{sucd(v)} + adjust_v$, $O^h = O^h + adjust_v$, $O^{h'} = O^{h'} - adjust_v$, $minrQ_k = minrQ_k - adjust_v$, and $F^m = F^m + adjust_v$, where $m = h, \ldots, h'-1$.
Step 1.7: If $s_v > 0$, set $h' = h' + 1$, $minrQ_k = \min(minrQ_k, Q_k - F^{h'})$, and go to Step 1.3.
Step 1.8: If $s_v = 0$, remove node v from H^+ and set $Temph' = h'$.
Step 1.9: If $H^+ \neq \varnothing$, go to Step 1.2.

In Step 2, the preceding nodes of $v \in H^+$ are checked as follows:

Step 2.1: Set $g = |H^+|$ and $Temph' = 0$.
Step 2.2: Set $v = H_g^+$, $l = Hl_g^+$, $h = SumN + l$, $h' = \max(SumN + l - 1, Temph')$, $pred(v) = r^{h'}$, and $minf_k = F^{h'}$.
Step 2.3: If $minf_k = 0$, set $g = g - 1$ and go to Step 2.2.
Step 2.4: If $h' \leqslant 0$, stop checking the preceding nodes of node v and go to Step 3.
Step 2.5: If $O^{h'} > 0$ and $pred(v) \notin V_H$, set $adjust_v = \min(s_v, minf_k, O^{h'})$.
Step 2.6: Set $s_v = s_v - adjust_v$, $s_{pred(v)} = s_{pred(v)} + adjust_v$, $O^h = O^h + adjust_v$, $O^{h'} = O^{h'} - adjust_v$, $minf_k = minf_k - adjust_v$, and $F^{g'} = F^{g'} - adjust_v$, where $g' = h', \ldots, h-1$.
Step 2.7: If $s_v > 0$, set $h' = h' - 1$, $minf_k = \min(minf_k, F^{h'-1})$, and go to Step 2.3.
Step 2.8: If $s_v = 0$, remove node v from H^+, and set $g = g - 1$ and $Temph' = h'$.
Step 2.9: If $H^+ \neq \varnothing$, go to Step 2.2.

Similar to Step 1, in Step 3, the succeeding nodes of $v \in H^+$ are checked as follows:

Step 3.1: Set $g = 1$ and $Temph' = 0$.
Step 3.2: Set $v = H_g^+$, $l = Hl_g^+$, $h = SumN + l$, $h' = \max(SumN + l + 1, Temph')$, $sucd(v) = r^{h'}$, and $minrQ_k = Q_k - F^h$.
Step 3.3: If $minrQ_k = 0$, set $g = g + 1$ and go to Step 3.2.
Step 3.4: If $h' \geqslant n_k + 1$, stop the adjustment procedure for the case $H^+ \neq \varnothing$.
Step 3.5: If $sucd(v) \in \overline{D}$ and $O^{h'} > 0$, set $adjust_v = \min(s_v, O^{h'}, minrQ_k)$.
Step 3.6: If $sucd(v) \in V_E$ and $O^{h'} > 0$, set $MaxAdjustT = MaxAdjustT + O^{h'}$.
Step 3.7: If $sucd(v) \in V_E$, set $adjust_v = \min(s_v, minrQ_k, MaxAdjustT)$ and update the total loading and unloading time and $MaxAdjustT$ by the Eqs. (32) and (33), respectively.
Step 3.8: Set $s_v = s_v - adjust_v$, $s_{sucd(v)} = s_{sucd(v)} + adjust_v$, $O^h = O^h + adjust_v$, $O^{h'} = O^{h'} - adjust_v$, $minrQ_k = minrQ_k - adjust_v$, and $F^{g'} = F^{g'} + adjust_v$, where $g' = h, \ldots, h'-1$.
Step 3.9: If $s_v > 0$, $h' = h' + 1$, $minrQ_k = \min(minrQ_k, Q_k - F^{h'})$, and go to Step 3.3.
Step 3.10: If $s_v = 0$, remove node v from H^+ and set $Temph' = h'$.
Step 3.11: If $H^+ \neq \varnothing$, go to Step 3.2.

3.6.1.2.2. For the case $H^- \neq \varnothing$. In this section, the loading/unloading operations at the nodes in $H^- \neq \varnothing$ are adjusted to make the infeasible solution feasible. The checking idea is that from the considered node $v \in H^-$ to the end of the route, it is possible to load more bikes or unload fewer bikes at the succeeding nodes of v to reduce the number of bikes loaded at node v.

The succeeding nodes of $v \in H^-$ are checked as follows:

Step 1: Set $g = 1$ and $Temph' = 0$.
Step 2: Set $v = H_g^-$, $l = Hl_g^-$, $h = SumN + l$, $h' = \max(SumN + l + 1, Temph')$, and $sucd(v) = r^{h'}$.
Step 3: If $h' \geqslant n_k + 1$, stop checking the succeeding nodes of node v.

Step 4: If $sucd(v) \notin V_H$ and $O^{h'} < 0$, set $MaxAdjustT = MaxAdjustT - O^{h'}$.
Step 5: If $sucd(v) \notin V_H$ and $s_{sucd(v)} > 0$, set $adjust_v = \min(s_v, s_{sucd(v)}, MaxAdjustT)$.
Step 6: Update the total loading and unloading time and *MaxAdjustT* by the following:

$$OprTime_k = OprTime_k + \begin{cases} 0, & \text{if } O^{h'} \geqslant 0 \\ (L+U)\cdot\min(adjust_v, |O^{h'}|), & \text{if } O^{h'} < 0 \end{cases} \tag{34}$$

$$MaxAdjustT = MaxAdjustT + \begin{cases} 0, & \text{if } O^{h'} \geqslant 0 \\ \min(adjust_v, |O^{h'}|), & \text{if } O^{h'} < 0 \end{cases} \tag{35}$$

Set $s_v = s_v + adjust_v$, $s_{sucd(v)} = s_{sucd(v)} - adjust_v$, $O^h = O^h - adjust_v$, $O^{h'} = O^{h'} + adjust_v$, and $F^{g'} = F^{g'} - adjust_v$, where $g' = h, \ldots, h'-1$.
Step 7: If $s_v < 0$, set $h' = h' + 1$ and go to Step 3.
Step 8: If $s_v = 0$, remove node v from H^- and set $Temph' = h'$.
Step 9: If $H^- \neq \emptyset$, go to Step 2, and stop otherwise.

3.6.2. Adjustment on route lengths

The length adjustment on a route considers the removal and insertion of nodes, denoted as *InsertionOfNodesR*( ) and *RemovalOfNodesR*( ), respectively. Subroutine 6 shows the framework of the adjustment on route lengths. The details of the operators are respectively introduced in Sections 3.6.2.1 and 3.6.2.2.

Subroutine 6 *RoutelengthAdj*( )

Input: A molecular structure ω, PE_ω, and vehicle k
1. Obtain $(\omega', PE_{\omega'}) = RemovalOfNodesR(\omega, PE_\omega, k)$
2. **if** $T - (TraTime_k + OprTime_k) > t'$
3. Obtain $(\omega', PE_{\omega'}) = InsertionOfNodesR(\omega', PE_{\omega'}, k)$
4. **end if**
5. **Output** ω' and $PE_{\omega'}$

Remarks: PE_ω is the evaluation function value of the solution ω. t' is the average travel time between each pair of nodes in the network, and it is calculated by $t' = \sum_{i,j \in V_0, i \neq j} t_{ij}/(|V_0|^2 - |V_0|)$.

3.6.2.1. Removal of nodes. *RemovalOfNodesR* is executed to remove node(s) (excluding the starting and ending depots) in a given route of vehicle k in ω, with the objective of maximizing the approximate reduction in the evaluation function value. The approximate maximal reduction *AMR* is calculated by $\min_{h=SumN+1,\ldots,SumN+n_k} Redu_h = \{Op + (t_{r^{h-1},r^{h+1}} - (t_{r^{h-1},r^h} + t_{r^h,r^{h+1}} + |O^h|\cdot(L+U)))\cdot\mu\}$, where $Op = \begin{cases} |O^h|, & r^h \in V_E \\ cP\cdot|O^h|, & r^h \in V_H \end{cases}$. If $AMR < 0$, the evaluation function value may be improved by removal, and therefore *RemovalOfNodesR* is performed and the node at $h' = \operatorname{argmin}_{SumN+1,\ldots,SumN+n_k} Redu_h$ is removed. Then, the evaluation function value of the new solution is calculated. If the evaluation function value of the new solution is improved, the new solution is kept and the removal operation continues until the evaluation function value cannot be further reduced; otherwise discarded the new solution and stop the procedure.

Once a node is removed from the route, the method proposed by Ho and Szeto (2014) is applied to obtain new feasible solutions. Suppose h' is the position of the node to be removed. O^{SumN}, $O^{h'-1}$, or $O^{h'+1}$ is adjusted. If $O^{h'} = 0$, $O^{h'}$ can be removed directly without changing O^{SumN}, $O^{h'-1}$, or $O^{h'+1}$. If $O^{h'} \neq 0$, the criteria for the adjustment listed in the first row of Table 2 (for $O^{h'} > 0$) and Table 3 (for $O^{h'} < 0$) are checked. Therefore, at most three feasible solutions can be obtained. One of the feasible solutions is randomly selected and treated as the resultant solution ω'. However, if none of the three criteria in the first row of Tables 2 and 3 is satisfied, the removal of nodes is not allowed to occur and the procedure stops.

Based on the change in O, the relevant variables are updated. If $O^{h'} = 0$, set $TraTime_k = TraTime_k + (t_{r^{h'-1},r^{h'+1}} - (t_{r^{h'-1},r^{h'}} + t_{r^{h'},r^{h'+1}}))$ and remove $F^{h'}$, $O^{h'}$, and $r^{h'}$. If $O^{h'} > 0$ or $O^{h'} < 0$, the applied adjustments are tabulated in Tables 2 and 3.

3.6.2.2. Insertion of nodes. Given a solution ω, the operator *InsertionOfNodes*( ) only considers adding node(s) to the route of vehicle k if $T - (TraTime_k + OprTime_k) > t'$, where $t' = \sum_{i,j \in V_0, i \neq j} t_{ij}/(|V_0|^2 - |V_0|)$. The following is the detailed procedure with four main steps:

Table 2
Adjustments applied for the case $O^{h'} > 0$ in the operators *RemovalOfNodesR* and *RemovalOfNodesT*.

	Adjusting O^{SumN} (at the starting depot)	Adjusting $O^{h'-1}$ (at the preceding node)	Adjusting $O^{h'+1}$ (at the succeeding node)
Criteria for adjustment	$O^{h'} \leqslant minrQ_k$ and $O^{h'} \leqslant s_0$ The above conditions allow that extra $O^{h'}$ bikes are loaded onto vehicle k at the starting depot without violating the vehicle capacity constraint	$O^{h'} \leqslant s_{r^{h'-1}}$ This criterion ensures that the final inventory at $r^{h'-1}$ is non-negative after the adjustment	$O^{h'} \leqslant s_{r^{h'+1}}$ This condition ensures that the final inventory at $r^{h'+1}$ is non-negative after the adjustment
Adjustment of current inventory, s_i*	$s_0 = s_0 - O^{h'}$ $s_{r^{h'}} = s_{r^{h'}} + O^{h'}$	$s_{r^{h'-1}} = s_{r^{h'-1}} - O^{h'}$ $s_{r^{h'}} = s_{r^{h'}} + O^{h'}$	$s_{r^{h'}} = s_{r^{h'}} + O^{h'}$ $s_{r^{h'+1}} = s_{r^{h'+1}} - O^{h'}$
Adjustment of bicycle flow array, F*	For $h = SumN, \ldots, h'-1$, $F^h = F^h + O^{h'}$ Remove $F^{h'}$	$F^{h'-1} = F^{h'-1} + O^{h'}$ Remove $F^{h'}$	Remove $F^{h'}$
Adjustment of $OprTime_k$*	$OprTime_k = OprTime_k$ (i.e., no need to update)	If $O^{h'-1} \geqslant 0$, $OprTime_k = OprTime_k$; If $O^{h'-1} < 0$, $OprTime_k = OprTime_k - (L+U)\cdot\min(\lvert O^{h'-1}\rvert, O^{h'})$	If $O^{h'+1} \geqslant 0$, $OprTime_k = OprTime_k$; If $O^{h'+1} < 0$, $OprTime_k = OprTime_k - (L+U)\cdot\min(\lvert O^{h'+1}\rvert, O^{h'})$
Adjustment of $TraTime_k$*	$TraTime_k = TraTime_k + (t_{r^{h'-1},r^{h'+1}} - (t_{r^{h'-1},r^{h'}} + t_{r^{h'},r^{h'+1}}))$.		
Adjustment of O*	$O^{SumN} = O^{SumN} + O^{h'}$ Remove $O^{h'}$	$O^{h'-1} = O^{h'-1} + O^{h'}$ Remove $O^{h'}$	$O^{h'+1} = O^{h'+1} + O^{h'}$ Remove $O^{h'}$
Adjustment of r*	Remove $r^{h'}$.		

Remarks:
(i) $minrQ_k$ is the minimum residual capacity of vehicle k before visiting node $r^{h'}$ and is obtained by $\min_{h=SumN,\ldots,SumN+h'-1}\{Q_k - F^h\}$
(ii) Only if the criteria of adjustment are satisfied, the adjustments with (*) can be applied.
(iii) $SumN$ is calculated by $SumN = \sum_{k'=1}^{k-1}(n_{k'} + 2)$.

Step 1: Randomly pick a position $g \in [1, n_k + 1]$ for inserting a node.
Step 2: Node v is a currently unbalanced node (i.e., $s_v - q_v \neq 0$) randomly selected from $Nearby_{pred(v),\varepsilon}$, where $pred(v) = r^{SumN+g-1}$ and $\varepsilon = \overline{k}_k$.
Step 3: Node v is inserted into position g if the insertion does not violate the time constraint (i.e., $TraTime_k + OprTime_k + (t_{pred(v),v} + t_{v,sucd(v)}) - t_{pred(v),sucd(v)} < T$), where $sucd(v) = r^{SumN+g}$. The adjustments applied to other related variables are listed below:
  Step 3.1: For $g' = n_k, n_k = 1, \ldots, g$, set $r^{g'+1} = r^{g'}$, $O^{g'+1} = O^{g'}$, $F^{g'+1} = F^{g'}$; set $r^g = v$, $O^g = 0$, and $F^g = F^{g-1}$.
  Step 3.2: Set $n_k = n_k + 1$.
  Step 3.3: Set $TraTime_k = TraTime_k + (t_{pred(v),v} + t_{v,sucd(v)}) - t_{pred(v),sucd(v)}$.
  Step 3.4: *OperationAdj* (Subroutine 5 introduced in Section 3.6.1) is executed to adjust the loading/unloading quantity at inserted node v.
  Step 3.5: Calculate the evaluation function value of the new solution ω'.
Step 4: If the evaluation function value of ω' is better than that of the original solution ω, ω' is kept and go to Step 1; otherwise, ω' is discarded and the procedure stops.

3.6.3. Route repairing

For each route, once the time constraint is violated, the operator *RemovalOfNodesT* is executed to remove the node associated with *AMR* until the travel time of the considered route no longer violates the time constraint. The ways of selecting nodes and adjusting relevant variables are the same as *RemovalOfNodesR* introduced in Section 3.6.2.1. However, the stopping criterion of the two operators is different: *RemovalOfNodesR* is executed until the objective value is no longer improved or none of the adjustment criteria (listed in the first row of Tables 2 and 3) can be satisfied; *RemovalOfNodesT* is executed until the total operational time of the route no longer violates the time constraint. If no feasible solution can be obtained by removing the node at $h' = \underset{h=SumN+1,\ldots,SumN+n_k}{\operatorname{argmin}} Redu_h$ and the total operational time is still longer than T, then set $Redu_{h'} = T$ and continue the procedure to remove node(s).

Table 3
Adjustments applied for the case $O^{h'} < 0$ in the operators *RemovalOfNodesR* and *RemovalOfNodesT*.

	Adjusting $O^{R\cdot(k-1)}$ (at the starting depot)	Adjusting $O^{h'-1}$ (at the preceding node)	Adjusting $O^{h'+1}$ (at the succeeding node)
Criteria for adjustment	$\lvert O^{h'}\rvert \leqslant minf_k$ and $s_0 - O^{h'} \leqslant c_0$ The above two conditions allow that fewer bikes are loaded onto vehicle k at the starting depot without violating any constraint	$s_{r^{h'-1}} - O^{h'} \leqslant c_{r^{h'-1}}$ This condition ensures that the final inventory at $r^{h'-1}$ is within its capacity after the adjustment	$s_{r^{h'+1}} - O^{h'} \leqslant c_{r^{h'+1}}$ The above criterion ensures that the capacity constraint at node $r^{h'+1}$ is not violated after the adjustment is applied
Adjustment of current inventory, s_i^*	Same as Table 2	Same as Table 2	Same as Table 2
Adjustment of bicycle flow array, F^*	Same as Table 2	Same as Table 2	Same as Table 2
Adjustment of $OprTime_k^*$	Same as Table 2	If $O^{h'-1} \geqslant 0$, $OprTime_k = OprTime_k - (L+U)\cdot\min(O^{h'-1}, \lvert O^{h'}\rvert)$; If $O^{h'-1} < 0$, $OprTime_k = OprTime_k$	If $O^{h'+1} \geqslant 0$, $OprTime_k = OprTime_k - (L+U)\cdot\min(O^{h'+1}, \lvert O^{h'}\rvert)$; If $O^{h'+1} < 0$, $OprTime_k = OprTime_k$
Adjustment of $TraTime_k^*$	Same as Table 2		
Adjustment of O^*	Same as Table 2	Same as Table 2	Same as Table 2
Adjustment of r^*	Same as Table 2		

Remarks:
(i) $minf_k$ is the minimum number of bikes on vehicle k before visiting node $r^{h'}$ and is obtained by $\min_{h=SumN,\ldots,SumN+h'-1}\{F^h\}$.
(ii) Only if the criteria of the adjustment are satisfied, the adjustments with (*) can be applied.
(iii) $SumN$ is calculated by $SumN = \sum_{k'=1}^{k-1}(n_{k'}+2)$.

3.6.4. 2-Opt

2-*Opt* is applied to a selected route obtained from the elementary reactions. This procedure considers the inversion of all possible sub-sequences (consisting of at least two nodes) excluding the starting and ending depots in the route. Once a route that satisfies the time constraint is obtained, the loading/unloading quantities are determined by *OperationDetermination* (Subroutine 8 introduced in Section 3.7). The best improved *feasible* solution found according to the evaluation function replaces ω, and the procedure stops. This is different from a traditional local search where the search continues until no improvement can be found.

3.6.5. Removal of adjacent repeated nodes

Given the route of vehicle k, *RemovalOfRepeatedN* is used to remove one of the identical adjacent nodes. Suppose $v = r^{SumN+l_1} = r^{SumN+l_2}$ ($l_1 + 1 = l_2$ and $0 \leqslant l_1 < l_2 \leqslant n_k$), node v at l_2 is removed from the route. The relevant variables, including the loading and unloading quantities, bike loads on vehicle k, and $OprTime_k$, are updated as follows:

(1) For the case $O^{SumN+l_1}\cdot O^{SumN+l_2} \geqslant 0$, either loading or unloading is operated at both l_1 and l_2, and no update of $OprTime_k$ is needed; for the case $O^{SumN+l_1}\cdot O^{SumN+l_2} < 0$, in total, fewer bikes are handled by the vehicle after combining O^{SumN+l_1} and O^{SumN+l_2}. Set $OprTime_k = OprTime_k - (L+U)\cdot\min(\lvert O^{SumN+l_1}\rvert, \lvert O^{SumN+l_2}\rvert)$.
(2) $O^{SumN+l_1} = O^{SumN+l_1} + O^{SumN+l_2}$ and remove O^{SumN+l_2};
(3) Remove F^{SumN+l_2};
(4) $n_k = n_k - 1$.

3.6.6. Implementation of solution adjustment

Subroutine 7 shows the framework of the solution adjustment and it is implemented once a new solution is obtained by reaction operators introduced in Section 3.5.

Subroutine 7 *SolutionAdjustment*( )

Input: A solution ω, PE_{ω}, and vehicle k

1. **if** $TraTime_k + OprTime_k > T$
2. $(\omega', PE_{\omega'}) = RemovalOfNodesT(\omega, PE_{\omega}, k)$
3. **if** $\mathrm{mod}(cHit'_{\omega}, OptStep) = 0$ **then**
4. $(\omega', PE_{\omega'}) = 2\text{–}Opt(\omega', PE_{\omega'}, k)$
5. **end if**
6. **end if**
7. **if** $\mathrm{mod}(cHit'_{\omega}, OptStep) = 0$ **then**
8. $(\omega', PE_{\omega'}) = 2\text{–}Opt(\omega', PE_{\omega'}, k)$
9. **end if**
10. $(\omega', PE_{\omega'}) = RemovalOfRepeatedN(\omega', PE_{\omega'}, k)$
11. $(\omega', PE_{\omega'}) = RoutelengthAdj(\omega', PE_{\omega'}, k)$
12. **Output** ω' and $PE_{\omega'}$

Remarks: $cHit'_{\omega}$ records the total number of consecutive iterations to undergo either on-wall ineffective collisions or intermolecular ineffective collisions. *OptStep* is a parameter that indicates the frequency of running 2-*Opt* in terms of the number of iterations undergo either on-wall ineffective collisions or the inter-molecular ineffective collisions. Lines 3 and 7 are not executed for the solutions obtained from the reactions of decomposition and synthesis (introduced in Section 3.5), i.e., 2-*Opt* is applied to the newly obtained solution(s) from decomposition or synthesis without any condition requirements. However, for the solutions obtained from on-wall ineffective collisions and inter-molecular ineffective collisions, the criterion in both Lines 3 and 7 must be satisfied to execute 2-*Opt*.

3.7. Calculation of loading and unloading quantities

For the solutions in 2-Opt (introduced in Section 3.6.4) or the reaction operators other than on-wall ineffective collision (introduced in Section 3.5), the loading/unloading quantities along the route of vehicle k need to be updated.

Given the current state of each node, the method for initial solution construction (i.e., Step 3.2 in Section 3.3) is used to calculate the loading and unloading quantities at each visited node. This procedure is denoted as *OperationCalculation*. The only difference between *OperationCalculation* and the initial solution construction heuristic is that in *OperationCalculation*, all nodes in the route are given and there is no need to add more to it.

Subroutine 8 shows the framework of how to calculate and adjust the loading and unloading quantities for a given route.

Subroutine 8 *OperationDetermination*( )

Input: A molecular structure ω and vehicle k

Table 4
Interpretations of the parameters used in the original CRO and the enhanced CRO.

Parameter	Interpretation
PopSize	The initial number of solutions in *Pop*
InitialKE	The initial KE value assigned to each solution in the initialization stage
KELossRate	Maximum fraction of the reduction in KE in on-wall ineffective collisions
MoleColl	Fraction of all elementary reactions corresponding to inter-molecular reactions
β	A parameter in the occurrence criterion of synthesis (a parameter that represents the least amount of KE in a molecule)
α	A parameter in the occurrence criterion of decomposition (a parameter that represents the maximum number of consecutive iterations to undergo either on-wall ineffective collisions or inter-molecular ineffective reactions if the current best solution is not updated)
MaxIteration	The maximum number of iterations.
*OptStep*ˆ	A parameter in the occurrence criterion of 2-*Opt* (a parameter that indicates the frequency of running 2-*Opt* in terms of the number of iterations undergo either on-wall ineffective collisions or inter-molecular ineffective collisions)

ˆ *OptStep* is only used in the enhanced CRO.

1. $\omega' = OperationCalculation(\omega, k)$
2. $\omega' = FlowBackRevise(\omega', k)$
3. $\omega' = DepotOperCalculation(\omega', k)$
4. $\omega' = OperationAdjHard(\omega', k)$
5. **Output** ω'

Remarks: The operator *OperationAdjHard* is used to adjust the loading/unloading quantities at all the visited nodes $i \in V_H$ (introduced in Section 3.6.1.2). Line 1 is not executed in the stage of initialization (Subroutine 9, introduced in Section 3.6.6) in the enhanced CRO.

3.8. Implementation of the enhanced CRO

The initialization is performed by **Subroutine 9**. The overall procedure of the enhanced CRO is described by **Algorithm 1**. The meanings of the parameters are also listed in Table 4.

Subroutine 9 *Initialization*( )

Input: Problem-specific information (evaluation function, constraints)
1. Assign parameter values
2. Generate *PopSize* initial feasible solutions.
3. **for** each solution ω **do**
4. Calculate PE_ω by setting it to be the evaluation function value $z'(\omega)$
5. $\omega' = OperationDetermination(\omega)$
6. Assign $KE_{\omega'}$ with the value of *InitialKE*
7. Assign $Hit_{\omega'} = 0$
8. **end for**
9. Let the central energy buffer be buffer and *buffer* = 0
10. Determine the current best solution among all the initial solutions
11. **Output** *Pop*

Remarks: Line 5 is not executed in the subroutine *Initialization* in the original CRO.

Algorithm 1 *Enhanced CRO*

1. $Pop = Initialization(\ )$
2. Let *iter* = 0
3. **while** $iter < maxIteration$ **do**
4. Get random *molet* in the interval (0,1)
5. **if** $molet > MoleColl$ **then**
6. Select a molecule M from *Pop* randomly
7. **if** $Hit'_\omega - \alpha > 0$ **then**
8. $(M'_1, M'_2, Success) = NewDecomposition(M, buffer)$
9. **if** *Success* = TRUE **then**
10. Remove M from *Pop*
11. Add M'_1 and M'_2 to *Pop*
12. $Hit'_{\omega'_1} = 0$ and $Hit'_{\omega'_2} = 0$
13. **end if**
14. **else**
15. $(M', buffer) = NewOnwallIneffCollision(M, buffer)$
16. **end if**
17. **else**
18. Select two molecules M_1 and M_2 from *Pop* randomly
19. **if** $KE_{\omega_1} < \beta$ and $KE_{\omega_2} < \beta$, and $PopSize > 2$ **then**
20. $(M', Success) = NewSynthesis(M_1, M_2)$
21. **if** *Success* = TRUE **then**
22. Remove M_1 and M_2 from *Pop*
23. Add M' to *Pop*
24. $Hit'_{\omega'} = 0$

25. **end if**
26. **else**
27. $(M_1', M_2') = NewInterMoleIneffCollision(M_1, M_2)$
28. **end if**
29. **end if**
30. $iter = iter + 1$
31. **end while**
32. **Output** the overall best solution and its evaluation function value

Similar to the original CRO, the enhanced version has four principle reactions. However, these reactions are different in the following:

1. For the enhanced CRO, *NewOnwall* is adopted in the on-wall ineffective collision to deal with both routes and loading/unloading quantities, whereas only the swapping operator is used to get new routes in the original CRO.
2. Unlike *NewDecom* in the enhanced CRO, a circular shift is applied to obtain new routes from the decomposition in the original CRO.
3. For intermolecular ineffective collisions, the swapping of two sub-sequences with equal length in a route is used in the enhanced CRO, whereas the swapping of any two nodes is applied to the original CRO to obtain new routes. Moreover, the calculation and adjustment of the loading/unloading quantities are considered in *NewInter*.
4. For synthesis, the enhanced CRO and the original CRO adopt the same method to handle vehicle routes, but only the enhanced CRO considers the adjustments on the loading/unloading quantities at visited nodes.

The following are also the differences between the original and enhanced CRO.

- In the enhanced CRO, the definition of Hit and decomposition criterion proposed by Szeto et al. (2016) are used, not those in Lam and Li (2010). In the enhanced CRO, Hit_ω in the decomposition criterion (line 7) refers to the total number of consecutive iterations for either on-wall ineffective collisions or the intermolecular ineffective collisions (i.e., the total number of searches among the neighbor solutions) before the current best solution is updated. For each molecular structure ω, if the current best solution is not updated after either an on-wall ineffective collision or an intermolecular ineffective collision occurs, Hit_ω increases by one; otherwise, Hit_ω is reset to zero. The decomposition criterion $Hit_\omega - \alpha > 0$ means that no better solution is obtained after searching the neighbor solutions of the solution ω α times. If such a criterion is met, the decomposition will be carried out to obtain a very different solution (non-neighbor). Hit_ω is set to zero after the decomposition or synthesis has occurred.
- After the occurrence of any reaction, the current best solution is updated if a new solution with a lower evaluation function value is obtained. This current best solution is the best value obtained by comparing all solutions in the pool. However, the original CRO determines the current best solution obtained by comparing all of the solutions generated from a specific solution.
- In the enhanced CRO, $cHit_\omega$ is introduced to record the total number of consecutive iterations to undergo either on-wall ineffective collisions or intermolecular ineffective collisions. Only if the decomposition or synthesis occurs, $cHit_\omega$ is set to zero; otherwise, it increases by one. To avoid getting many replicated solutions from 2-*Opt*, "$\mathrm{mod}(cHit_\omega, OptStep) = 0$" is added to control the occurrence of 2-*Opt* to the new neighbor solutions obtained from on-wall ineffective collisions and intermolecular ineffective reactions. It means that 2-*Opt* is performed in every *OptStep* iterations for on-wall ineffective collisions and the intermolecular ineffective collisions.
- In the enhanced CRO, the condition ($PopSize > 2$) is added to the synthesis criterion (line 19) to avoid *Pop* being too small to allow any reaction, especially an inter-molecular ineffective collision, in the subsequent iteration.

4. Numerical studies

To tune our proposed heuristic and illustrate its efficiency and accuracy, three sets of instances were used, which are available at http://web.hku.hk/~ceszeto/instances.zip. The first set of instances of different sizes were created based on the instance sets used by Ho and Szeto (2017), which was generated based on the largest bike-sharing system in the United States, *Citi Bike* in New York. The details of instance sets are listed in Table 5. For the instance of $|V| = 60$, 60 stations were randomly selected from 302 stations and the depot in the file name *n302* at https://sites.google.com/site/drsinho/instances/hlns-set3.zip. Two nodes were randomly selected as depots. 60×0.4 nodes are hardly accessed nodes and 60×0.6 nodes are easily accessed nodes. Other instances were generated

Table 5
Instance set based on Citi Bike.

$\|V\|$	Percentage of hardly accessed nodes ($\|V_h\|/\|V\| \times 100\%$)	Instances
60	40%	n302
90	40%	n302
300	40%	n471
400	40%	n471

Table 6
Details of the instance set based on SAB.

$\|V\|$	$\|D\|$	$\|V_E\|$	$\|V_H\|$
42	1	33	9
78	1	57	21
95	1	65	30

similarly. In these instances, the travel times between nodes (in seconds) are asymmetric. The demand at each node is derived based on the penalty cost, i.e., the inventory that gives the lowest penalty cost. The time required for loading/unloading a bike is 60 s.

The second set of instances was created based on those used by Pal and Zhang (2017). It was generated based on the FFBSS operated in the University of South Florida's Tampa campus named Share-A-Bull (SAB). Each instance contains one depot, easily accessed nodes, and hardly accessed nodes. The nodes whose demand is zero and capacity is relatively low were selected as the hardly accessed nodes. The details of the second set of instances are listed in Table 6.

The third instance was derived from the instance of 30 nodes provided by Rainer-Harbach et al. (2015), which was in turn generated based on 2011 real-world data provided by Citybike Vienna. Besides the depot, 11 nodes were randomly selected from those 30 nodes. One, six and four of these 11 nodes were randomly selected as the second depot, easily accessed nodes, and hardly accessed nodes, respectively.

Two to three vehicles with a capacity of 10 or 20 bikes are deployed for the repositioning operation. Different combinations of vehicle capacities in a fleet are considered. For $|K| = 2$, three combinations are considered, including (10, 10), (10, 20), and (20, 20); For $|K| = 3$, four combinations are considered, including (10, 10, 10), (10, 10, 20), (10, 20, 20), and (20, 20, 20).

The weight $\mu = 0.00001$, which implies that the main objective is to minimize total unsatisfied demand. The heuristic was written in C++, compiled by Visual Studio, and ran on a computer with an Intel i7-3770 CPU @ 3.4 GHz and 32 GB RAM.

4.1. Parameter tuning

In this section, the parameters listed in Table 4 are tuned in the following order: *InitialKE*, β, *PopSize*, *MoleColl*, *KELossRate*, α, and *MaxIteration*. Among them, *InitialKE* and β control the ability to get new solutions and have a direct and close relationship with the PE of the problem (Szeto et al., 2014). Thus, as in their study, these values were tuned based on the initial objective value, denoted as *InitialObj* $= \sum_{i\in V_E}(\max(q_i - s_i^0, 0)) + \sum_{i\in V_H} cP \cdot s_i^0$ (i.e., the total number of unsatisfied customers before repositioning is performed plus the total penalty assigned to the bikes parked at all nodes $i \in V_H$). The trial values for tuning the ratios of these two parameters were the *multiples* of *InitialObj*. As in most of the existing studies, twenty random seeds were used. The average of the best objective values from 20 runs determined the parameter values.

The Citi Bike instance used for parameter tuning has 60 stations. Two vehicles were used to perform repositioning. The capacity of each vehicle is 10. The repositioning duration is 9000 s. The loading or unloading time for a bicycle is 60 s. The initial objective value (*InitialObj*) is 302. The initial parameter values were as follows: *InitialKE* = 302; β = 302; *PopSize* = 30; *MoleColl* = 0.5; *KELossRate* = 0.5; α = 300; and *MaxIteration* = 10,000. *InitialKE* was first tuned. It took on different values from the set {151, 302, ...,1812} whereas the other parameter values remained unchanged. According to the first plot of Fig. 8, the best average result was found when *InitialKE* = 755. Then, $\beta \in$ {151, 302, ...,1510} was tuned with an *InitialKE* of 755 and the remaining parameters were kept at their initial values. According to the second plot of Fig. 8, the best average result was found when β = 453. This procedure was repeated for other parameters.

Other tuning results are plotted in Fig. 8. The parameter values are set as follows: *InitialKE* = 755 (i.e., 2.5 × *InitialObj*); β = 453 (i.e., 1.5 × *InitialObj*); *PopSize* = 30; *MoleColl* = 0.3; *KELossRate* = 0.2; α = 400; and *MaxIteration* = 13,000. These parameter values were applied in the following numerical experiments, except for *InitialKE* and β, in which the deduced ratios of 2.5 and 1.5 were used. Regarding the value of *MoleColl*, it implies that about 70% of the overall reactions are uni-molecular reactions (i.e., either decomposition or on-wall ineffective collision). This also implies that the operators for the decomposition and on-wall ineffective collision reaction are more efficient than those for the synthesis and inter-molecular ineffective collision.

4.2. Comparison between CPLEX and the enhanced CRO

To illustrate the effectiveness of the enhanced CRO, the comparisons of the performance of the enhanced CRO and CPLEX were conducted. For the first experiment, SAB instances were used. Two repositioning durations (T = 4,500 s and T = 9,000 s), two vehicle types ($\overline{Q}_1 = 10$ and $\overline{Q}_2 = 20$), two fleet sizes ($|K| = 2$ and $|K| = 3$), and all the combinations of vehicle capacities were considered. For each scenario (i.e., a given combination of an instance, a repositioning duration, a fleet size, and vehicle capacities) and each seed, the enhanced CRO was executed with a *maxIteration* value of 13,000. The best and average of the best solutions from 20 runs were used to evaluate their performance. These scenarios were also solved with IBM ILOG CPLEX 12.63 with the default settings. A running time of 2 h was imposed on CPLEX. However, CPLEX only checks the time at certain points. Therefore, some of the computing times reported here are longer than 2 h. The optimality tolerance of CPLEX was set to 0.01%. Thus, if the optimal gap, Gap_{opt}, is less than 0.01%, CPLEX gives an "optimal" objective value; otherwise, CPLEX only gives an upper bound on the objective value of an optimal solution.

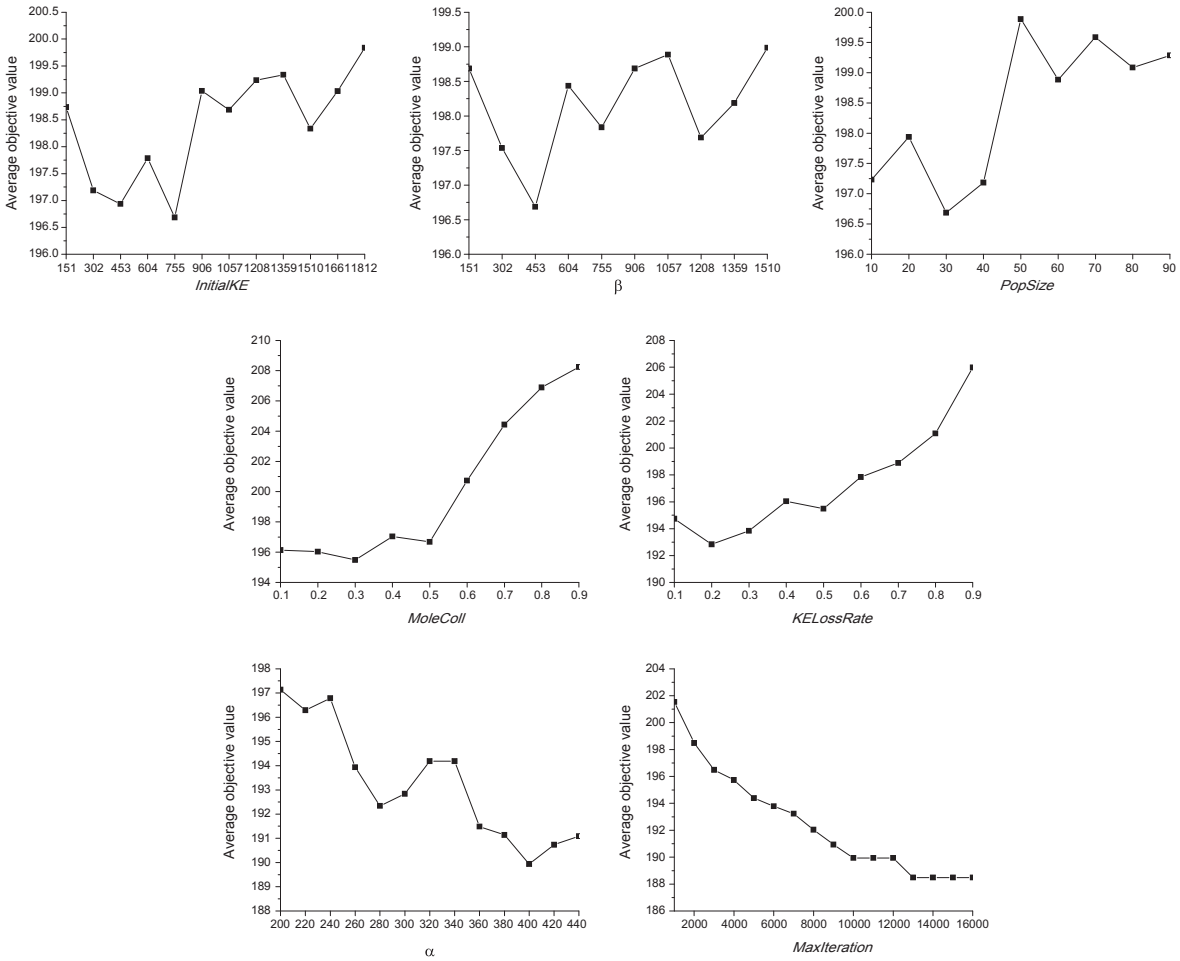

Fig. 8. Parameter tuning for the enhanced CRO.

Table 7
A comparison of the results between CPLEX and the enhanced CRO (instances based on SAB, $|K| = 2$ and $T = 4{,}500$ s).

$\|V\|$	Vehicle case	CPLEX			Enhanced CRO			Gap_1	Gap_2
		UB/Opt.	$Gap_{Opt.}$	CPU	Best	Average	CPU		
42	10, 10	47.07810	19.65	7200.29	41.08622	42.78785	5.73	−12.73	−9.11
	10, 20	47.07892	21.10	7200.23	38.10348	43.03868	6.59	−19.06	−8.58
	20, 20	46.07884	17.87	7200.23	41.09360	43.23765	5.68	−10.82	−6.17
78	10, 10	115.07872	71.54	7200.29	86.08172	93.88233	4.14	−25.20	−18.42
	10, 20	100.09061	68.95	7200.38	86.08172	94.23210	4.07	−14.00	−5.85
	20, 20	98.08020	61.18	7200.23	86.08172	93.88231	4.13	−12.23	−4.28
95	10, 10	159.07966	75.57	7200.90	133.07854	148.03071	6.56	−16.34	−6.95
	10, 20	161.08002	78.17	7200.77	133.08118	149.73177	6.33	−17.38	−7.05
	20, 20	159.07834	78.74	7200.32	140.07914	146.08161	5.39	−11.94	−8.17
Avg				7200.40			5.40	−15.52	−8.29

Tables 7–10 show the results obtained by CPLEX and the enhanced CRO under different repositioning durations, fleet size, capacities, and numbers of stations. Gap_1 and Gap_2 (in percent) indicate the performance of the enhanced CRO relative to that of CPLEX based on the averages of the best and average objective values, respectively. The CPU (in seconds) denotes the average computing time of 20 runs for the enhanced CRO or the computing time for CPLEX. From the results, we observe that CPLEX cannot obtain any optimal solutions with a 2-h time limit, and can only provide an upper bound for the problem.

Unlike the enhanced CRO, CPLEX cannot obtain feasible solutions in many cases when the repositioning duration becomes longer, the size of the instance becomes larger, or more vehicles are considered. For example, for the case $|K| = 2$ and $T = 4{,}500$ s (see

Table 8
A comparison of the results between CPLEX and the enhanced CRO (instances based on SAB, $|K| = 3$ and $T = 4{,}500$ s).

$\|V\|$	Vehicle case	CPLEX			Enhanced CRO			Gap_1	Gap_2
		UB/Opt.	$Gap_{Opt.}$	CPU	Best	Average	CPU		
42	10, 10, 10	39.12136	29.53736	7200.24	29.15882	34.14185	5.31	−25.47	−12.73
	10, 10, 20	41.12320	33.96897	7200.24	28.17058	33.54634	5.80	−31.50	−18.42
	10, 20, 20	43.12396	37.49066	7200.19	29.17004	33.99615	5.88	−32.36	−21.17
	20, 20, 20	46.11774	41.65552	7200.13	27.19572	33.94494	5.24	−41.03	−26.40
78	10, 10, 10	149.13393	93.88670	7200.19	63.12480	76.17304	6.68	−57.67	−48.92
	10, 10, 20	149.12523	93.88630	7200.29	63.12480	75.17276	6.81	−57.67	−49.59
	10, 20, 20	149.13054	93.88656	7201.77	63.12480	75.32262	6.98	−57.67	−49.49
	20, 20, 20	–	–	7200.24	70.11562	78.12230	6.72	–	–
95	10, 10, 10	–	–	7200.43	115.12138	124.67074	7.01	–	–
	10, 10, 20	–	–	7200.59	115.12138	124.67074	6.98	–	–
	10, 20, 20	–	–	7200.59	108.12218	120.82158	7.87	–	–
	20, 20, 20	–	–	7200.28	108.11820	123.56887	7.43	–	–
Avg				7200.43			6.56	−53.51	−43.60

Table 9
A comparison of the results between CPLEX and the enhanced CRO (instances based on SAB, $|K| = 2$ and $T = 9{,}000$ s).

$\|V\|$	Vehicle case	CPLEX			Enhanced CRO			Gap_1	Gap_2
		UB/Opt.	$Gap_{Opt.}$	CPU	Best	Average	CPU		
42	10, 10	–	–	7200.49	12.16932	14.51672	11.50	–	–
	10, 20	81.16892	99.79	7200.34	13.16960	14.96387	9.91	−83.78	−81.56
	20, 20	–	–	7200.35	11.16534	13.26514	11.83	–	–
78	10, 10	–	–	7200.65	22.17136	26.91653	13.89	–	–
	10, 20	–	–	7200.60	23.16744	27.21810	13.73	–	–
	20, 20	–	–	7200.65	22.17290	26.01462	13.83	–	–
95	10, 10	–	–	7200.77	53.16420	59.51566	13.08	–	–
	10, 20	–	–	7201.35	56.14816	60.21612	12.56	–	–
	20, 20	–	–	7200.80	53.15608	59.86439	12.55	–	–
Avg				7200.67			12.54	−83.78	−81.56

Table 10
A comparison of the results between CPLEX and the enhanced CRO (instances based on SAB, $|K| = 3$ and $T = 9{,}000$ s).

$\|V\|$	Vehicle case	CPLEX			Enhanced CRO			Gap_1	Gap_2
		UB/Opt.	$Gap_{Opt.}$	CPU	Best	Average	CPU		
42	10, 10, 10	–	–	7200.37	5.24622	6.89244	13.06	–	–
	10, 10, 20	–	–	7200.29	2.25912	5.79637	13.19	–	–
	10, 20, 20	80.26082	99.68602	7200.49	3.25192	6.29109	13.09	−95.95	−92.16
	20, 20, 20	–	–	7200.69	2.24228	5.64692	13.00	–	–
78	10, 10, 10	–	–	7200.27	12.24298	15.83869	17.87	–	–
	10, 10, 20	–	–	7201.35	12.24298	16.09286	17.81	–	–
	10, 20, 20	–	–	7201.27	11.24318	16.33620	17.69	–	–
	20, 20, 20	–	–	7200.15	12.24606	15.49208	17.57	–	–
95	10, 10, 10	–	–	7201.85	25.25544	30.39860	15.88	–	–
	10, 10, 20	–	–	7201.69	19.24142	30.44708	15.93	–	–
	10, 20, 20	–	–	7201.66	28.22136	31.49918	15.83	–	–
	20, 20, 20	–	–	7201.69	26.24524	32.54856	15.49	–	–
Avg				7200.98			15.53	−95.95	−92.16

Table 7), CPLEX can obtain feasible solutions for all the considered scenarios, while for the case $|K| = 2$ and $T = 9{,}000$ s (see Table 9), a feasible solution can only be found for 1 scenario. For the results obtained for the case $|K| = 3$ and $T = 4{,}500$ s (see Table 8), fewer feasible solutions are obtained by CPLEX compared with the results shown in Table 7. When only considering the results for the case $|K| = 3$ and $T = 4{,}500$ s (see Table 8), CPLEX cannot obtain feasible solutions for the scenarios related to $|V| = 95$. In contrast, the enhanced CRO always obtains good feasible solutions much faster than CPLEX for larger, longer repositioning time, or more vehicle

Table 11
A comparison of the results between CPLEX and the enhanced CRO ($|V| = 10$ and $T = 6{,}000$ s).

Vehicle case	CPLEX			Enhanced CRO			Gap_1	Gap_2
	UB/Opt.	$Gap_{Opt.}$	CPU	Best	Average	CPU		
10, 10	14.11220	0.00	398.05	14.11220	14.11304	6.06	0.00	0.01
20, 20	14.11220	0.00	914.32	14.11220	14.11304	5.96	0.00	0.01
Avg			656.19			6.01	0.00	0.01

Table 12
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 2$, $T = 9{,}000$ s, and the stopping criterion is generating the same number of solutions).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10	231.17015	250.21927	0.13	112.17280	116.97547	35.16	−51.48	−53.25
	10, 20	232.17761	251.22100	0.12	108.17486	111.02654	31.88	−53.41	−55.81
	20, 20	222.17529	240.87149	0.11	102.17672	105.87640	29.83	−54.01	−56.04
90	10, 10	453.17589	471.32520	0.16	291.17548	299.67495	39.88	−35.75	−36.42
	10, 20	452.17770	469.37751	0.13	290.17796	298.62567	34.84	−35.83	−36.38
	20, 20	446.17831	457.77790	0.13	292.17041	300.57581	32.60	−34.52	−34.34
300	10, 10	1326.17617	1335.67047	0.19	1115.17683	1137.37516	41.27	−15.91	−14.85
	10, 20	1311.17839	1334.77393	0.17	1124.17174	1142.57451	36.07	−14.26	−14.40
	20, 20	1314.17790	1331.17372	0.16	1130.17829	1148.07498	32.84	−14.00	−13.75
400	10, 10	1743.17776	1767.22281	0.29	1525.17791	1535.42420	54.72	−12.51	−13.12
	10, 20	1751.16993	1763.02691	0.24	1531.17682	1542.27630	45.81	−12.56	−12.52
	20, 20	1742.17788	1757.87657	0.21	1530.17834	1544.02659	41.18	−12.17	−12.17
Avg				0.17			38.01	−28.87	−29.42

Table 13
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 3$, $T = 9{,}000$ s, and the stopping criterion is generating the same number of solutions).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10, 10	219.24978	231.00229	0.20	77.26660	81.96296	40.25	−64.76	−64.52
	10, 10, 20	211.24247	229.94898	0.19	71.26161	75.81382	37.01	−66.27	−67.03
	10, 20, 20	210.26669	228.56069	0.18	66.25963	69.51339	35.81	−68.49	−69.59
	20, 20, 20	212.26654	227.61173	0.17	65.25746	68.06247	34.59	−69.26	−70.10
90	10, 10, 10	435.24627	452.55847	0.22	246.26301	250.76392	42.06	−43.42	−44.59
	10, 10, 20	423.26672	447.95853	0.20	236.26507	244.36407	40.12	−44.18	−45.45
	10, 20, 20	437.25621	447.76301	0.19	234.26618	240.36462	37.40	−46.42	−46.32
	20, 20, 20	427.26660	446.51397	0.19	232.26132	236.91253	36.76	−45.64	−46.94
300	10, 10, 10	1311.23481	1320.20781	0.27	1037.26274	1053.21149	45.64	−20.89	−20.22
	10, 10, 20	1294.25494	1316.35426	0.25	1043.25561	1054.51197	41.85	−19.39	−19.89
	10, 20, 20	1295.26680	1314.41071	0.24	1024.26178	1053.96332	39.91	−20.92	−19.81
	20, 20, 20	1297.26122	1311.31338	0.23	1030.25930	1051.91333	38.75	−20.58	−19.78
400	10, 10, 10	1731.25747	1749.90701	0.35	1434.26095	1448.36306	56.98	−17.15	−17.23
	10, 10, 20	1718.26687	1745.81137	0.32	1431.26238	1451.96377	50.94	−16.70	−16.83
	10, 20, 20	1720.26505	1746.30814	0.30	1441.26225	1452.76297	48.31	−16.22	−16.81
	20, 20, 20	1725.25861	1743.36614	0.29	1435.26232	1453.36146	44.95	−16.81	−16.63
Avg				0.24			41.96	−37.32	−37.61

scenarios.

Regarding the results obtained for a fixed repositioning duration and number of vehicles, the average computing time of the enhanced CRO varies from 5.40 to 15.53 s, while the average computing time of CPLEX is around 7,200 s. Moreover, all negative values of Gap_1 and Gap_2 indicate that the enhanced CRO obtains a better upper bound for each of the considered scenarios. Therefore, the enhanced CRO could obtain better solutions much faster than CPLEX.

For the results obtained under a fixed repositioning duration, the absolute values of the average Gap_1 and Gap_2 values obtained by

Table 14
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 2$, $T = 18{,}000$ s, and the stopping criterion is generating the same number of solutions).

$\|V\|$	Vehicle Case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10	207.35761	222.84449	0.39	63.32259	63.39512	96.94	−69.46	−71.55
	10, 20	208.35762	220.38612	0.36	63.30038	63.32182	89.55	−69.62	−71.27
	20, 20	207.35763	214.35133	0.31	63.28383	63.30535	79.29	−69.48	−70.47
90	10, 10	438.35761	449.70071	0.46	219.35821	224.00496	113.46	−49.96	−50.19
	10, 20	423.35762	441.39746	0.44	209.35491	213.85554	107.89	−50.55	−51.55
	20, 20	419.35763	439.68853	0.36	202.35983	202.37438	90.41	−51.75	−53.97
300	10, 10	1296.35765	1312.92968	0.61	958.35598	971.40337	134.31	−26.07	−26.01
	10, 20	1267.33377	1305.29833	0.56	953.35427	968.95299	126.21	−24.77	−25.77
	20, 20	1271.35772	1295.90764	0.55	944.35900	961.15427	125.74	−25.72	−25.83
400	10, 10	1714.35271	1737.08615	0.85	1346.35943	1361.05461	170.25	−21.47	−21.65
	10, 20	1719.35761	1739.83848	0.76	1355.35202	1366.10346	153.10	−21.17	−21.48
	20, 20	1698.35761	1734.10548	0.72	1344.35718	1363.95545	144.01	−20.84	−21.35
Avg				0.53			119.26	−41.74	−42.59

Table 15
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 3$, $T = 18{,}000$ s, and the stopping criterion is generating the same number of solutions).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10, 10	207.43801	212.70927	0.43	63.32035	63.33574	74.66	−69.48	−70.22
	10, 10, 20	208.53642	213.71197	0.40	63.29693	63.31964	67.53	−69.65	−70.37
	10, 20, 20	207.53642	213.44371	0.36	63.28080	63.30380	57.92	−69.51	−70.34
	20, 20, 20	207.53642	212.96663	0.32	63.28446	63.29512	52.07	−69.51	−70.28
90	10, 10, 10	418.46600	429.85526	0.63	202.41757	202.42999	109.82	−51.63	−52.91
	10, 10, 20	418.53641	429.40236	0.60	202.38466	202.40230	102.19	−51.64	−52.86
	10, 20, 20	420.53640	430.65388	0.57	202.36742	202.38281	95.35	−51.88	−53.01
	20, 20, 20	422.53641	427.51873	0.53	202.35983	202.37438	90.41	−52.11	−52.66
300	10, 10, 10	1257.53649	1286.42262	0.97	790.53080	814.58124	164.39	−37.14	−36.68
	10, 10, 20	1247.53651	1281.91774	0.91	798.52652	814.72886	154.03	−35.99	−36.44
	10, 20, 20	1264.53646	1282.28451	0.90	768.53003	803.58047	153.28	−39.22	−37.33
	20, 20, 20	1256.53650	1281.83386	0.85	777.52425	800.68165	150.36	−38.12	−37.54
400	10, 10, 10	1679.53645	1712.16385	1.21	1199.53702	1213.23127	197.47	−28.58	−29.14
	10, 10, 20	1690.53645	1709.11237	1.12	1187.53293	1210.78296	183.52	−29.75	−29.16
	10, 20, 20	1685.48204	1713.82547	1.07	1178.52748	1202.18112	175.62	−30.08	−29.85
	20, 20, 20	1673.53641	1700.02278	1.04	1181.53555	1201.58037	178.69	−29.40	−29.32
Avg				0.74			125.46	−47.11	−47.38

the enhanced CRO for the scenarios with $|K| = 3$ are always larger than those for the scenarios with $|K| = 2$. For example, when $T = 4{,}500$ s, the average values of Gap_1 and Gap_2 are respectively −15.52% and −8.29% for the case $|K| = 2$, while they are respectively −53.51% and −43.60% for the case $|K| = 3$. This implies that the enhanced CRO performs better than CPLEX when solving the FFBRP with more vehicles involved.

For the results of a constant fleet size, the absolute values of the average Gap_1 and Gap_2 values obtained by the enhanced CRO for the scenarios with long repositioning durations are larger than those with short repositioning durations. For example, when $|K| = 2$, $|V| = 42$, and the vehicle case is (10, 20), Gap_1 and Gap_2 are respectively −19.06% and −8.58% for $T = 4{,}500$ s but −83.78% and −81.56% for $T = 9{,}000$ s. This implies that the enhanced CRO performs better than CPLEX when solving the FFBRP with a longer repositioning duration.

Table 11 presents a comparison of the results between CPLEX and the enhanced CRO for the second experiment using the small Vienna instance. This comparison aims to prove that the enhanced CRO can obtain optimal solutions. The repositioning duration is set to 6,000 s. For the scenarios considered in Table 11, the solutions obtained by CPLEX are optimal. From Table 11, the enhanced CRO can also obtain optimal solutions with an average computational time of 6.01 s, which is much shorter than that of CPLEX (656.19 s). These results show that the enhanced CRO can find optimal solutions much faster than CPLEX.

Table 16
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 2$, $T = 9{,}000$ s, and the stopping criterion is the same running time).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10	226.17883	247.81964	35.16	16107289.00	112.17280	116.97547	35.16	−50.41
	10, 20	230.17782	247.72454	31.88	14601486.00	108.17486	111.02654	31.88	−53.00
	20, 20	222.17529	236.27303	29.83	13949974.00	102.17672	105.87640	29.83	−54.01
90	10, 10	444.17496	466.62278	39.88	17380840.00	291.17548	299.67495	39.88	−34.45
	10, 20	450.15999	466.57317	34.84	15096475.00	290.17796	298.62567	34.84	−35.54
	20, 20	443.17814	455.17648	32.60	14316050.00	292.17041	300.57581	32.60	−34.07
300	10, 10	1325.17411	1333.62199	41.27	13948831.00	1115.17683	1137.37516	41.27	−15.85
	10, 20	1311.17839	1332.37431	36.07	11740718.00	1124.17174	1142.57451	36.07	−14.26
	20, 20	1311.17816	1326.07392	32.84	10871203.00	1130.17829	1148.07498	32.84	−13.80
400	10, 10	1743.17776	1764.47220	54.72	16071682.00	1525.17791	1535.42420	54.72	−12.51
	10, 20	1749.17651	1762.27436	45.81	13384879.00	1531.17682	1542.27630	45.81	−12.46
	20, 20	1742.17788	1757.32660	41.18	12824482.00	1530.17834	1544.02659	41.18	−12.17
Avg				38.01				38.01	−28.54

Table 17
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 3$, $T = 9{,}000$ s, and the stopping criterion is the same running time).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10, 10	219.24978	228.90130	40.25	13555299.00	77.26660	81.96296	40.25	−64.76
	10, 10, 20	211.24247	228.04426	37.01	13081337.00	71.26161	75.81382	37.01	−66.27
	10, 20, 20	210.23110	227.35990	35.81	12642521.00	66.25963	69.51339	35.81	−68.48
	20, 20, 20	209.26659	221.36451	34.59	12445609.00	65.25746	68.06247	34.59	−68.82
90	10, 10, 10	433.22606	447.75224	42.06	13834971.00	246.26301	250.76392	42.06	−43.16
	10, 10, 20	423.26672	445.95679	40.12	13500666.00	236.26507	244.36407	40.12	−44.18
	10, 20, 20	425.26654	445.41055	37.40	12394650.00	234.26618	240.36462	37.40	−44.91
	20, 20, 20	427.26660	442.81591	36.76	12353212.00	232.26132	236.91253	36.76	−45.64
300	10, 10, 10	1298.23987	1317.60505	45.64	11958994.00	1037.26274	1053.21149	45.64	−20.10
	10, 10, 20	1294.25494	1311.95668	41.85	10976166.00	1043.25561	1054.51197	41.85	−19.39
	10, 20, 20	1291.24456	1310.55910	39.91	10711198.00	1024.26178	1053.96332	39.91	−20.68
	20, 20, 20	1295.25490	1309.11460	38.75	10271538.00	1030.25930	1051.91333	38.75	−20.46
400	10, 10, 10	1728.25791	1744.95074	56.98	13315916.00	1434.26095	1448.36306	56.98	−17.01
	10, 10, 20	1713.25535	1744.81095	50.94	11934522.00	1431.26238	1451.96377	50.94	−16.46
	10, 20, 20	1716.24918	1743.70686	48.31	11659193.00	1441.26225	1452.76297	48.31	−16.02
	20, 20, 20	1719.26648	1740.16309	44.95	10625775.00	1435.26232	1453.36146	44.95	−16.52
Avg				41.96				41.96	−37.05

4.3. Comparison between the enhanced CRO and the original CRO

To compare the performance of the performance of the enhanced CRO and the original CRO, various Citi Bike instances with sizes ranged from 60 to 400 nodes were considered. Two repositioning durations ($T = 9{,}000$ s and $T = 18{,}000$ s), two vehicle capacities ($\overline{Q}_1 = 10$ and $\overline{Q}_2 = 20$), and two fleet sizes ($|K| = 2$ and $|K| = 3$) were considered.

To ensure a fair comparison, for each scenario and each seed, the enhanced CRO was first executed with a *maxIteration* value of 13,000. The number of solutions generated by the enhanced CRO and the computational time were then recorded and set as the stopping criterion for the original CRO. The best and average of the best solutions from 20 runs were used to evaluate their performance.

Tables 12–15 show the results comparison between the original CRO and the enhanced CRO with the stopping criterion of generating the same number of solutions, and Tables 16–19 show the results of the original CRO and the enhanced CRO with the same running time. Gap_1 and Gap_2 (in percent) indicate the performance of the enhanced CRO relative to that of the original version based on the averages of the best and average objective values, respectively. The CPU (in seconds) denotes the average computing time of 20 runs for the enhanced or original CRO. From Tables 12–15, all Gap_1 and Gap_2 values are negative, meaning that the solution quality of the enhanced CRO is better than that of the original version. Under a fixed fleet size, when the repositioning duration becomes longer, the average values of $|Gap_1|$ and $|Gap_2|$ become larger. For example, for the case $|K| = 2$, the average values of Gap_1

Table 18
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 2$, $T = 18{,}000$ s, and the stopping criterion is the same running time).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10	207.35761	220.06308	96.94	63.32259	63.39512	96.94	−69.46	−71.19
	10, 20	207.35761	217.47063	89.55	63.30038	63.32182	89.55	−69.47	−70.88
	20, 20	207.35763	213.90307	79.29	63.28383	63.30535	79.29	−69.48	−70.40
90	10, 10	429.30226	443.03326	113.46	219.35821	224.00496	113.46	−48.90	−49.44
	10, 20	422.35760	439.69026	107.89	209.35491	213.85554	107.89	−50.43	−51.36
	20, 20	419.35762	434.53997	90.41	202.35983	202.37438	90.41	−51.75	−53.43
300	10, 10	1282.33069	1309.82265	134.31	958.35598	971.40337	134.31	−25.26	−25.84
	10, 20	1263.35761	1300.05077	126.21	953.35427	968.95299	126.21	−24.54	−25.47
	20, 20	1271.35772	1293.25198	125.74	944.35900	961.15427	125.74	−25.72	−25.68
400	10, 10	1704.28059	1733.87347	170.25	1346.35943	1361.05461	170.25	−21.00	−21.50
	10, 20	1703.31647	1735.88836	153.10	1355.35202	1366.10346	153.10	−20.43	−21.30
	20, 20	1698.35761	1730.85554	144.01	1344.35718	1363.95545	144.01	−20.84	−21.20
Avg				119.26			119.26	−41.44	−42.31

Table 19
A comparison of the results between the original CRO and the enhanced CRO ($|K| = 3$, $T = 18{,}000$ s, and the stopping criterion is the same running time).

$\|V\|$	Vehicle case	Original CRO			Enhanced CRO			Gap_1	Gap_2
		Best	Average	CPU	Best	Average	CPU		
60	10, 10, 10	207.36406	210.90667	74.66	63.32035	63.33574	74.66	−69.46	−69.97
	10, 10, 20	207.33490	210.77735	67.53	63.29693	63.31964	67.53	−69.47	−69.96
	10, 20, 20	207.45118	210.51747	57.92	63.28080	63.30380	57.92	−69.50	−69.93
	20, 20, 20	207.44039	210.19923	52.07	63.28446	63.29512	52.07	−69.49	−69.89
90	10, 10, 10	418.45288	428.48960	109.82	202.41757	202.42999	109.82	−51.63	−52.76
	10, 10, 20	418.47718	426.74761	102.19	202.38466	202.40230	102.19	−51.64	−52.57
	10, 20, 20	418.53642	429.41324	95.35	202.36742	202.38281	95.35	−51.65	−52.87
	20, 20, 20	419.53713	426.15917	90.41	202.35983	202.37438	90.41	−51.77	−52.51
300	10, 10, 10	1257.53649	1283.60878	164.39	790.53080	814.58124	164.39	−37.14	−36.54
	10, 10, 20	1235.50515	1277.41008	154.03	798.52652	814.72886	154.03	−35.37	−36.22
	10, 20, 20	1243.49265	1276.01464	153.28	768.53003	803.58047	153.28	−38.20	−37.02
	20, 20, 20	1229.53642	1278.18156	150.36	777.52425	800.68165	150.36	−36.76	−37.36
400	10, 10, 10	1679.53645	1706.49943	197.47	1199.53702	1213.23127	197.47	−28.58	−28.91
	10, 10, 20	1690.53645	1706.11659	183.52	1187.53293	1210.78296	183.52	−29.75	−29.03
	10, 20, 20	1685.48204	1709.92272	175.62	1178.52748	1202.18112	175.62	−30.08	−29.69
	20, 20, 20	1667.53649	1696.97619	178.69	1181.53555	1201.58037	178.69	−29.14	−29.19
Avg				125.46			125.46	−46.85	−47.15

and Gap_2 are respectively −28.87% and −29.42% if $T = 9{,}000$ s, while they are respectively −41.74% and −42.59% if $T = 18{,}000$ s. Under a fixed repositioning duration, when the fleet size becomes larger, the absolute values of the average Gap_1 and Gap_2 values become larger. Here, we take $T = 9{,}000$ s as an example. If $|K| = 2$, the average values of Gap_1 and Gap_2 respectively are −28.87% and −29.42%, whereas if $|K| = 3$, the average values of Gap_1 and Gap_2 are respectively −37.32% and −37.61%. This indicates that the enhanced CRO is more effective than the original CRO to solve multi-vehicle FFBRPs with a larger fleet size. Nevertheless, under a fixed repositioning duration, a fixed number of vehicles, and a fixed capacity combination, both Gap_1 and Gap_2 become less negative when the size of the instance is larger.

The results shown in Tables 16–19 indicate the same trends as the results in Tables 12–15. With longer running time, the results obtained by the original CRO in Tables 16–19 are slightly improved compared to the results obtained by the original CRO in Tables 12–15. However, all gap values are still negative. Therefore, these reflect that the enhanced CRO is more effective than the original CRO to solve multi-vehicle BRPs with a longer repositioning duration, more vehicles, and smaller instances.

4.4. The effectiveness of introducing OperationAdj into the enhanced CRO

In our study, the operator *OperationAdj* (refer to Section 3.6.1), which considers the characteristics of the FFBRP, is introduced to improve the solution quality obtained by the proposed heuristic. In this section, the discussed scenarios for the instance $|V| = 400$

Table 20
A comparison of the results of the enhanced CRO with and without the operator *OperationAdj* ($|V| = 400$ and the stopping criterion is generating the same number of solutions).

T (h)	$\|K\|$	Vehicle case	Enhanced CRO without *OperationAdj*			Enhanced CRO			Gap
			Best	Average	CPU	Best	Average	CPU	
2.5	2	10, 10	1631.15254	1647.35954	39.45	1525.17791	1535.42420	54.72	−7.29
		10, 20	1626.17743	1649.46032	32.85	1531.17682	1542.27630	45.81	−6.95
		20, 20	1622.17743	1655.76265	31.67	1530.17834	1544.02659	41.18	−7.24
2.5	3	10, 10, 10	1565.26539	1589.35483	38.28	1434.26095	1448.36306	56.98	−9.73
		10, 10, 20	1565.25036	1595.15372	33.40	1431.26238	1451.96377	50.94	−9.86
		10, 20, 20	1570.26349	1596.65165	32.76	1441.26225	1452.76297	48.31	−9.90
		20, 20, 20	1573.25695	1595.15093	31.13	1435.26232	1453.36146	44.95	−9.76
5	2	10, 10	1507.31935	1526.87703	93.74	1346.35943	1361.05461	170.25	−12.18
		10, 20	1501.29484	1525.93543	90.86	1355.35202	1366.10346	153.10	−11.70
		20, 20	1483.35451	1517.33385	84.85	1344.35718	1363.95545	144.01	−11.25
5	3	10, 10, 10	1378.52554	1415.51453	117.29	1199.53702	1213.23127	197.47	−16.67
		10, 10, 20	1384.53608	1409.15560	106.34	1187.53293	1210.78296	183.52	−16.38
		10, 20, 20	1390.52401	1413.21801	103.19	1178.52748	1202.18112	175.62	−17.55
		20, 20, 20	1379.53191	1406.71620	107.71	1181.53555	1201.58037	178.69	−17.07
Avg					67.39			110.40	−11.68

Table 21
A comparison of the results of the enhanced CRO with and without the operator *OperationAdj* ($|V| = 400$ and the stopping criterion is the same running time).

T (h)	$\|K\|$	Vehicle case	Enhanced CRO without *OperationAdj*			Enhanced CRO			Gap
			Best	Average	CPU	Best	Average	CPU	
2.5	2	10, 10	1631.15254	1647.35938	54.72	1525.17791	1535.42420	54.72	−7.29
		10, 20	1626.17743	1649.46009	45.81	1531.17682	1542.27630	45.81	−6.95
		20, 20	1622.17743	1655.76265	41.18	1530.17834	1544.02659	41.18	−7.24
2.5	3	10, 10, 10	1565.26539	1589.35483	56.98	1434.26095	1448.36306	56.98	−9.73
		10, 10, 20	1565.25036	1595.15372	50.94	1431.26238	1451.96377	50.94	−9.86
		10, 20, 20	1570.26349	1596.65165	48.31	1441.26225	1452.76297	48.31	−9.90
		20, 20, 20	1573.25695	1595.15025	44.95	1435.26232	1453.36146	44.95	−9.76
5	2	10, 10	1507.31935	1526.87703	170.25	1346.35943	1361.05461	170.25	−12.18
		10, 20	1501.29484	1525.93543	153.10	1355.35202	1366.10346	153.10	−11.70
		20, 20	1483.35451	1517.33385	144.01	1344.35718	1363.95545	144.01	−11.25
5	3	10, 10, 10	1378.52554	1415.51453	197.47	1199.53702	1213.23127	197.47	−16.67
		10, 10, 20	1384.53608	1409.15560	183.52	1187.53293	1210.78296	183.52	−16.38
		10, 20, 20	1390.52401	1413.21801	175.62	1178.52748	1202.18112	175.62	−17.55
		20, 20, 20	1379.53191	1406.71620	178.69	1181.53555	1201.58037	178.69	−17.07
Avg					110.40			110.40	−11.68

(described in Section 4.3) were also solved by the enhanced CRO without *OperationAdj* to illustrate the effectiveness of introducing this operator into the heuristic. The best and average of the best solutions from 20 runs were used to evaluate the performance of the considered heuristic. To ensure a fair comparison, for each scenario and each seed, the enhanced CRO without *OperationAdj* was executed to generate the same number of solutions as the enhanced CRO (discussed in Section 4.3).

The results are shown in Table 20. Gap (in percent) herein refers to the difference in the average of the best objective values of the concerned scenario obtained from 20 runs by the enhanced CRO and that without *OperationAdj*, normalized by the average objective value obtained from the enhanced CRO. The value of Gap implies the relative improvement (or deterioration) on the objective value of the enhanced CRO to that of the enhanced CRO without *OperationAdj*. If it is negative, the performance of the enhanced CRO is improved by the introduction of *OperationAdj*; it is weakened otherwise. CPU denotes the average running time in seconds.

From Table 20, the average running time of the enhanced CRO without *OperationAdj* (67.39 s) is much shorter than that of the enhanced CRO (110.40 s). Since the difference of the running time between the enhanced CRO without *OperationAdj* and the enhanced CRO (with *OperationAdj*) is large, the enhanced CRO without *OperationAdj* was run again with the stopping criterion as the same running time as the enhanced CRO. The results are shown in Table 21. Comparing the results shown in Tables 20 and 21, the gap values are the same, which indicate that there is no significant improvement in the results obtained from the enhanced CRO without *OperationAdj* even the heuristic runs longer.

From Tables 20 and 21, the Gap values for all discussed scenarios are negative, and the average Gap value is −11.63%. This

implies that although the implementation of the operator *OperationAdj* takes time, the general performance of the enhanced CRO is improved by incorporating the operator *OperationAdj* into the CRO to solve the FFBRP. The absolute value of Gap becomes larger when T or/and fleet size increase. This indicates that the operator *OperationAdj* provides competitive advantages to the enhanced CRO to obtain solutions of multiple-vehicle FFBRPs with long repositioning durations. The operator *OperationAdj* aims to adjust the loading/unloading quantities at nodes along the route, meet the demand at easily accessed nodes, and reduce the penalty for the bikes at hardly accessed nodes to the maximum extent, thereby improving solution quality. It can be concluded that the enhanced CRO proposed in this paper should incorporate the operator *OperationAdj* to improve its solution quality.

5. Conclusion

In this paper, a static free-floating bike repositioning problem with multiple heterogeneous vehicles, multiple depots, and multiple visits is proposed. An enhanced version of CRO is introduced to solve this problem. It incorporates a concept of the nearby-node set to narrow the search space and encapsulates an operator to adjust the loading and unloading quantities at visited nodes by making use of the node characteristics. The computation experiments were conducted on instances up to 400 nodes and 2 depots. The computational results demonstrate that the enhanced CRO gets better solutions than the original CRO and CPLEX for all discussed scenarios and has potential to tackle the repositioning problem for larger, longer repositioning duration, and more vehicle instances. The results also prove that the incorporation of the loading and unloading quantity adjustment procedure is beneficial.

It is noted that in our formulation, the number of nodes highly depends on the number of bikes in the system. The number of nodes can be large if there are many bikes in the system. To reduce the number of nodes considered, one can form a cluster for those bicycles nearby and a node is used to represent a cluster. Moreover, this study only introduced an enhanced CRO to solve the problem. The effectiveness of this heuristic compared with traditional meta-heuristics, such as variable neighborhood search, tabu search, and genetic algorithm, is not known. This comparison is left to future studies.

Acknowledgements

This research was supported by a grant from the National Natural Science Foundation of China (71771194). The authors are grateful to the three reviewers for their constructive comments.

References

Abdullah, Z., 2017. Bike-sharing: Users share perks, gripes. The Straits Times, 26 March 2017 < http://www.straitstimes.com/singapore/bike-sharing-users-share-perks-gripes > .

Alvarez-Valdes, R., Belenguer, J.M., Benavent, E., Bermudez, J.D., Muñoz, F., Vercher, E., 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., Robinet, L., 2011. Balancing the stations of a self service “bike hire” system. RAIRO – Operat. Res. 45 (1), 37–61.

Brinkmann, J., Ulmer, M.W., Mattfeld, D.C., 2015a. Inventory routing for bike sharing systems. Working Paper (2015-01-12) < https://www.tu-braunschweig.de/Medien-DB/winfo/publications/wp_brinkmann_inventory_routing_bike_sharing.pdf > .

Brinkmann, J., Ulmer, M.W., Mattfeld, D.C., 2015b. Short-term strategies for stochastic inventory routing in bike sharing systems. In: Proceedings of the 18th EURO Working Group on Transportation, Transportation Research Procedia, vol. 10, pp. 364–373.

Caggiani, L., Ottomanelli, M., 2012. A modular soft computing based method for vehicles repositioning in bike-sharing systems. Proc. – Social Behav. Sci. 54, 675–684.

Caggiani, L., Ottomanelli, M., Camporeale, R., Binetti, M., 2016. Spatio-temporal clustering and forecasting method for free-floating bike sharing systems. In: Świątek, J., Tomczak, J.M. (Eds.), Advances in Systems Science: Proceedings of the International Conference on Systems Science 2016 (ICSS 2016). Springer International Publishing, pp. 244–254.

Casazza, M., 2016. Exactly solving the split pickup and split delivery vehicle routing problem on a bike-sharing system. Retrieved from: http://hal.upmc.fr/hal-01304433.

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

Contardo, C., Morency, C., Rousseau, L.-M., 2012. Balancing A Dynamic Public Bike-Sharing System, Technical Report CIRRELT 2012. Montréal.

Cruz, F., Subramanian, A., Bruck, B.P., Iori, M., 2017. A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. Comp. Operat. Res. 79, 19–33.

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

Dell’Amico, M., Iori, M., Novellani, S., Stützle, T., 2016. A destroy and repair algorithm for the bike sharing rebalancing problem. Comp. Operat. Res. 71, 149–162.

Di Gaspero, L., Rendl, A., Urli, T., 2013a. Constraint-based approaches for balancing bike sharing systems. In: Schulte, C. (Ed.), Principles and Practice of Constraint Programming. Springer Berlin Heidelberg, pp. 758–773.

Di Gaspero, L., Rendl, A., Urli, T., 2013b. A hybrid ACO + CP for balancing bicycle sharing systems. In: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics. Springer Berlin Heidelberg, pp. 198–212.

Di Gaspero, L., Rendl, A., Urli, T., 2016. Balancing bike sharing systems with constraint programming. Constraints 21 (2), 318–348.

Erdoğan, G., Battarra, M., Wolfler Calvo, R., 2015. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. Euro. J. Operat. Res. 245 (3), 667–679.

Erdoğan, G., Laporte, G., Wolfler Calvo, R., 2014. The static bicycle relocation problem with demand intervals. Euro. J. Operat. Res. 238 (2), 451–457.

Forma, I.A., Raviv, T., Tzur, M., 2015. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Transport. Res. Part B: Methodol. 71, 230–247.

Ho, S.C., Szeto, W.Y., 2014. Solving a static repositioning problem in bike-sharing systems using iterated tabu search. Transport. Res. Part E: Logist. Transport. Rev. 69, 180–198.

Ho, S.C., Szeto, W.Y., 2017. A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. Transport. Res. Part B: Methodol. 95, 340–363.

Kadri, A.A., Kacem, I., Labadi, K., 2016. A branch-and-bound algorithm for solving the static rebalancing problem in bicycle-sharing systems. Comp. Indust. Eng. 95, 41–52.

Labadi, K., Benarbia, T., Barbot, J.P., Hamaci, S., Omari, A., 2015. Stochastic Petri net modeling, simulation and analysis of public bicycle sharing systems. IEEE Trans. Autom. Sci. Eng. 12 (4), 1380–1395.

Lam, A.Y.S., Li, V.O.K., 2010. Chemical-reaction-inspired metaheuristic for optimization. IEEE Trans. Evolution. Comput. 14 (3), 381–399.

Li, Y., Szeto, W.Y., Long, J.C., Shui, C.S., 2016. A multiple type bike repositioning problem. Transport. Res. Part B 90, 263–278.
Meddin, R., DeMaio, P., 2017. The Bike-Sharing World Map < http://www.bikesharingworld.com/ > (access on 28 November 2017).
Nair, R., Miller-Hooks, E., 2011. Fleet management for vehicle sharing operations. Transport. Sci. 45 (4), 524–540.
Nair, R., Miller-Hooks, E., Hampshire, R.C., Bušić, A., 2013. Large-scale vehicle sharing systems: analysis of Vélib'. Int. J. Sustain. Transport. 7 (1), 85–106.
Pal, A., Zhang, Y., 2017. Free-floating bike sharing: solving real-life large-scale static rebalancing problems. Transport. Res. Part C: Emerg. Technol. 80, 92–116.
Papazek, P., Raidl, G.R., Rainer-Harbach, M., Hu, B., 2013. A PILOT/VND/GRASP hybrid for the static balancing of public bicycle sharing systems. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (Eds.), Computer Aided Systems Theory – EUROCAST 2013. Springer Berlin Heidelberg, pp. 372–379.
Papazek, P., Kloimüllner, C., Hu, B., Raidl, G.R., 2014. Balancing bicycle sharing systems: an analysis of path relinking and recombination within a GRASP hybrid. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (Eds.), International Conference on Parallel Problem Solving from Nature. Springer International Publishing, pp. 792–801.
Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R., 2013. Balancing bicycle sharing systems: A variable neighborhood search approach. In: European Conference on Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg, pp. 121–132.
Rainer-Harbach, M., Papazek, P., Raidl, G.R., Hu, B., Kloimüllner, C., 2015. PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. J. Glob. Optim. 63 (3), 597–629.
Raviv, T., Tzur, M., Forma, I., 2013. Static repositioning in a bike-sharing system: models and solution approaches. EURO J. Transport. Logist. 2 (3), 187–229.
Regue, R., Recker, W., 2014. Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem. Transport. Res. Part E: Logist. Transport. Rev. 72, 192–209.
Reiss, S., Bogenberger, K., 2016. Validation of a relocation strategy for Munich's bike sharing system. Transport. Res. Proc. 19, 341–349.
Rudloff, C., Lackner, B., 2014. Modeling demand for bikesharing systems: neighboring stations as source for demand and reason for structural breaks. Transport. Res. Rec.: J. Transport. Res. Board 2430, 1–11.
Schuijbroek, J., Hampshire, R.C., van Hoeve, W.-J., 2017. Inventory rebalancing and vehicle routing in bike sharing systems. Euro. J. Operat. Res. 257 (3), 992–1004.
Singhvi, D., Singhvi, S., Frazier, P.I., Henderson, S.G., O'Mahony, E., Shmoys, D.B., Woodard, D.B., 2015. Predicting bike usage for New York City's bike sharing system. In: AAAI 2015 Workshop on Computational Sustainability.
Shui, C.S., Szeto, W.Y., 2018. Dynamic green bike repositioning problem – a hybrid rolling horizon artificial bee colony algorithm approach. Transport. Res. Part D: Transp. Environ. 60, 119–136.
Szeto, W.Y., Wang, Y., Wong, S.C., 2014. The chemical reaction optimization approach to solving the environmentally sustainable network design problem. Comput.-Aided Civil Infrastruct. Eng. 29 (2), 140–158.
Szeto, W.Y., Liu, Y., Ho, S.C., 2016. Chemical reaction optimization for solving a static bike repositioning problem. Transport. Res. Part D: Transp. Environ. 47, 104–135.
Zhang, D., Yu, C., Desai, J., Lau, H.Y.K., Srivathsan, S., 2017. A time-space network flow approach to dynamic repositioning in bicycle sharing systems. Transport. Res. Part B: Methodol. 103, 188–207.