Transportation Research Part D 47 (2016) 104–135

Contents lists available at ScienceDirect

Transportation Research Part D

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

Chemical reaction optimization for solving a static bike repositioning problem

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

[a] The University of Hong Kong Shenzhen Institute of Research and Innovation, Shenzhen, China
[b] Department of Civil Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong
[c] Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, Building 2628, 320, 8210 Aarhus V, Denmark

ARTICLE INFO

Article history:

ABSTRACT

In this paper, the single-vehicle static repositioning problem is studied. The objective of repositioning is to minimize the weighted sum of unmet customer demand and operational time on the vehicle route. To solve this problem, chemical reaction optimization (CRO) is proposed to handle the vehicle routes, and a subroutine is proposed to determine the loading and unloading quantities at each visited station. An enhanced version of CRO is proposed to improve the solution quality of the original CRO by adding new operators, rules, and intensive neighbor solution search methods. The concept of a neighbor-node set is proposed to narrow the solution search space. To illustrate the efficiency and accuracy of the enhanced CRO, different test scenarios are set and the results obtained from IBM ILOG CPLEX, the original CRO, and the enhanced CRO are compared. The computational results indicate that the enhanced CRO provides high-quality solutions with shorter computing times than those of IBM ILOG CPLEX and provides better solutions than the original CRO. The results also demonstrate that incorporation of the two neighbor-node sets into the enhanced CRO improves the solution quality, and the probability of running the intensive search should increase with iteration in the final part of the main stage of the algorithm to obtain better solutions.

Introduction

Bicycle riding is a green mode of transportation. It is an alternative to the use of private cars (Fürst, 2014) and can be used as a feeder mode to public transportation by serving as the last-mile mode for passengers. This mode has been researched widely, including, but not limited to, bike safety (e.g., Bai et al., 2013), cyclist choice behavior (e.g., Caulfield et al., 2012), bicycle commuting (e.g., Buehler, 2012; Thigpen et al., 2015), e-bikes (e.g., Fyhri and Fearnley, 2015), bike network design (e.g., Lu, 2016), bike network flow analysis (e.g., Kitthamkesorn et al., 2016), and bike sharing (e.g., Fishman et al., 2014a; Castillo-Manzano et al., 2015).

Bike sharing is currently very popular worldwide. As of 12 April 2016, public bike-sharing systems were available in about 1019 cities and included approximately 1,324,530 bicycles around the world (Meddin and DeMaio, 2015). Bike sharing can enhance the shift to modes other than motor vehicles and the use of public transport and can reduce CO_2 emissions and other pollutant emissions from motorized traffic to improve air quality. Ricci (2015) reported on user surveys worldwide that 2% of private car usage has shift to bicycles in London (London BCH), 7% in Lyon, France (Vélo'v), 9% in Washington, D.C., United States

* Corresponding author.
E-mail address: ceszeto@hku.hk (W.Y. Szeto).

http://dx.doi.org/10.1016/j.trd.2016.05.005
1361-9209/

(Capital Bikeshare), and 19% in Melbourne, Australia (Melbourne Bike Share). The overall distance traveled by motor vehicles has been reduced by bike sharing in most of the investigated cities except for London, even when the additional vehicle usage introduced by repositioning the bikes is considered (Fishman et al., 2014a). In London (Goodman and Cheshire, 2014) and Washington, D.C. (Shaheen et al., 2014), the use of bike sharing to rail stations was significantly higher, and in Paris (Shaheen et al., 2014), the use to Metro stations was higher. However, an analysis of the use of bike sharing in Melbourne revealed that the number of trips was significantly higher for docking stations located in areas with relatively less accessible public transit opportunities, which suggests that bike sharing can be a potential substitute for public transport in addition to connecting to it (Fishman et al., 2014b). It is therefore important to conduct bike-sharing research to examine its true benefits and effects.

A common phenomenon of bike-sharing is that the number of bikes required at some stations is insufficient to satisfy the user demand while other stations have an excess of bicycles. Bicycles are thus moved from stations with excess bikes to those with insufficient bikes to lower the total unmet demand. This operational problem is called a bike repositioning problem.

There are two types of bike repositioning problems: user-based and vehicle-based. User-based repositioning problems refer to cases in which price incentives are provided to customers to encourage them to return their bicycles to nearby underused stations. Very few papers (Fricker and Gast, 2014; Pfrommer et al., 2014; Ruch et al., 2014; Singla et al., 2015) have studied the user-based repositioning problem; most existing studies deal with vehicle-based repositioning problems. In these problems, vehicles are deployed to redistribute bicycles among stations. These problems are also known as bike-sharing rebalancing problems (BRPs) (Dell'Amico et al., 2014) and require the determination of vehicle routes and loading and unloading quantities at each visited station. If only one vehicle is deployed to operate bicycle repositioning, the resultant bike repositioning problem can be viewed as a special one-commodity pickup-and-delivery capacitated vehicle routing problem studied by Hernández-Pérez and Salazar-González (2004) and Hernández-Pérez and Salazar-González (2007).

In general, there are two types of vehicle-based repositioning problems: static and dynamic. If the operation is performed at night, the number of bicycles in each station and the number each station requires are often known before repositioning takes place and remain unchanged during the repositioning operation. This problem is referred to as a static bike repositioning problem. A dynamic repositioning problem considers changes in the number of bicycles in each station and the number required in each station over time. This type of repositioning is always operated during the day. To the best of our knowledge, very few studies (Caggiani and Ottomanelli, 2012; Contardo et al., 2012; Regue and Recker, 2014; Brinkmann et al., 2015a, 2015b; Labadi et al., 2015) have examined the dynamic bike repositioning problem, and most existing studies deal with the static bike repositioning problem.

Previous studies of vehicle-based repositioning problems have considered different measures in the objective function, including (1) the total travel time (cost) of the vehicle (Benchimol et al., 2011; Chemla et al., 2013; Di Gaspero et al., 2013a; Dell'Amico et al., 2014); (2) the total operational cost of repositioning, including the vehicles' travel time and the operational time for loading and unloading (Raviv et al., 2013; Di Gaspero et al., 2013a; Papazek et al., 2013; Rainer-Harbach et al., 2015); (3) the total absolute deviation from the target number of bicycles at each station (Di Gaspero et al., 2013a, 2013b; Papazek et al., 2013; Rainer-Harbach et al., 2015); (4) the total unmet demand for bicycles and lockers (Contardo et al., 2012); (5) the sum of the squared deviations between the target and final inventory levels over all stations (Brinkmann et al., 2015a); (6) the sum of the relocation cost and lost user cost (Caggiani and Ottomanelli, 2012); (7) the total penalty cost at all stations (Raviv et al., 2013; Ho and Szeto, 2014; Forma et al., 2015); and (8) the expected number of due date violations (Brinkmann et al., 2015b). The objectives are normally determined according to the concerns of the bike-sharing operator.

Vehicle-based repositioning problems can be formulated with various operational constraints. For example, the routing constraint proposed by Benchimol et al. (2011) forced the deployed vehicle to visit each station exactly once. However, Raviv et al. (2013) introduced a routing constraint that did not require all stations to be visited. They also limited the repositioning duration by the repositioning time constraint. Benchimol et al. (2011) and Chemla et al. (2013) did not consider the repositioning time constraint but added another constraint that required the inventory level of each station equal to its target level after repositioning. Meanwhile, Nair and Miller-Hooks (2011) introduced a probabilistic level-of-service constraint such that the repositioning activity was required to satisfy a certain proportion of all near-term demand scenarios in the planning horizon but ignored the routing constraint. The operational constraints included in the problems should be related to the application. These constraints determine the complexity of the problems.

Exact methods, such as branch-and-cut algorithms (see Dell'Amico et al., 2014; Erdoğan et al., 2014, 2015), can be used to solve vehicle-based repositioning problems. However, it is more complicated to solve a vehicle-based bike repositioning problem than a vehicle routing problem (VRP) because the former further considers loading and unloading quantities at each node. The bike repositioning problem is also an NP-hard problem because a VRP is already NP-hard. It is intractable to use exact methods to solve large, realistic repositioning problems. Previous studies (e.g., Raviv et al., 2013; Ho and Szeto, 2014) have also illustrated this point by conducting numerical experiments. Hence, most of the existing literature focuses on the development of inexact methods to obtain good solutions with short computing times.

A brief summary of inexact solution methods are as follows.

- Approximation method
  - 9.5-approximation algorithm (Benchimol et al., 2011)
- Heuristics or metaheuristics
  - Cluster-first route-second (Schuijbroek et al., 2013)
  - Iterated tabu search (Ho and Szeto, 2014)

  - PILOT and variable neighborhood descent (Kloimüllner et al., 2014; Rainer-Harbach et al., 2015)
  - GRASP and variable neighborhood descent (Kloimüllner et al., 2014; Rainer-Harbach et al., 2015)
  - Variable neighborhood search (VNS) (Kloimüllner et al., 2014; Rainer-Harbach et al., 2015)
  - GRASP hybridized with path relinking (Papazek et al., 2014)
- Hybrid heuristic and exact methods
  - branch-and-cut algorithm with tabu search (Chemla et al., 2013)
  - 3-step math heuristic (Forma et al., 2015)

Most of these inexact methods do not consider problem properties and station characteristics. For example, in reality, not all stations need to be visited by the repositioning vehicles, for several reasons. First, the stations at which the demand for bikes equals the supply do not require extra bikes when the objective is to minimize unmet demand, neither should the bikes at those stations be taken away; thus it is not necessary to visit these "balanced" stations. Second, some stations may not be reached by the vehicles due to their short operational time. Third, it is not necessary for the trucks to visit all stations that have more bikes than required (i.e., pickup stations), especially when the objective is to minimize the total unmet demand and the total supply from all pickup stations is greater than the total demand from stations that have fewer bikes than required (i.e., drop-off stations), Fourth, the pickup stations may not have sufficient total supply for the drop-off stations. In such a case, even if the trucks visit all drop-off stations, the total demand from all drop-off stations cannot be satisfied. Hence, it is unwise to visit all drop-off stations. Only Ting and Liao (2013) and Ho and Szeto (2014, 2016) considered the characteristics of the stations to narrow the solution search space and to develop efficient heuristics to solve their problems. They classified stations into pickup and drop-off stations and made use of the station characteristic in problem-solving.

The preceding heuristics applied to BRPs are mainly classical heuristics. Many recently developed heuristics have not been used to solve BRPs. These recently developed heuristics have been applied to many applications with great success. For example, the chemical reaction optimization algorithm (CRO) is a newly proposed population-based metaheuristic that mimics the interactions between molecules in a reaction (Lam and Li, 2010). The total number of solutions kept simultaneously by the algorithm may change from time to time (Lam and Li, 2010). CRO has the ability to avoid getting the search stuck at local minima. Unlike other heuristics, CRO allows the diversification and intensification of solutions to occur automatically rather than using a fixed sequence of operators for these purposes. Moreover, as indicated by Lam and Li (2010), CRO may be considered an optimization algorithm that allows users to use their favorable heuristic components for specific optimization problems, owing to the changeable components of CRO, including the criteria and mechanisms of various operators. The extents of intensification and diversification in the solution search are controlled easily by the operators. Hence, this meta-heuristic can be applied to a wide range of optimization problems and has already been proven to perform well in solving classic NP-hard problems, such as the quadratic assignment problem, the resource-constrained project scheduling problem, and the channel assignment problem (e.g., Lam and Li, 2010). It also has had wide application in various fields, such as the fuzzy rule learning problem (e.g., Lam et al., 2012), sensor deployment for air pollution monitoring (e.g., Yu et al., 2012), and stock portfolio selection (e.g., Xu et al., 2011). However, the performance of CRO in solving BRPs is unknown. Therefore, we are interested in improving CRO and testing the performance of the improved algorithm to solve our studied problem.

In this paper, the (single-vehicle) static repositioning problem is studied and a modified version of the formulation proposed by Raviv et al. (2013) is developed. The objective of repositioning is to minimize the weighted sum of the unmet customer demand and the operational time on the vehicle's route, which is different from the BRPs studied in the literature. Hence, existing methods to solve BRPs cannot be directly applied to our problem. CRO is proposed to handle the vehicle route, and a subroutine is proposed to determine the loading and unloading quantities at each visited station. To make CRO suitable for the solution of our static repositioning problem, an enhanced version of CRO is proposed. New operators and neighbor solution search methods are added to the original CRO. Two concepts of neighbor-node sets associated with each station are proposed to narrow the solution search space. A candidate station belongs to at least one of the two neighbor-node sets of another station based on the deviation from the demand at the candidate station and the operational time of an arc between them (which includes the travel time along the arc and the expected time for loading or unloading bicycles at the candidate station). This concept provides competitive advantages to CRO in the solution of static repositioning problems with a large size. To illustrate the efficiency and accuracy of the enhanced CRO, different test scenarios are set up, and the results obtained from IBM ILOG CPLEX, the original CRO, and the enhanced CRO are compared. In particular, the performance of the newly introducing features to CRO is demonstrated.

The contributions of this paper are as follows.

1. This paper proposes a new heuristic for the solution of a static BRP. The proposed heuristic makes use of the problem property (e.g., the objective function and station characteristic) to solve the loading and unloading subproblem. It is different from and more efficient than other methods (e.g., Rainer-Harbach et al., 2015) that rely on solving the subproblem as a linear programming or max-flow subproblem in each iteration. The solution space is reduced with the use of the proposed two new neighbor-node sets.
2. This paper shows that CRO can be used to quickly obtain good solutions to large routing problems. To the best of our knowledge, it is one of the first applications of CRO to routing problems with great success and the first for BRPs.
3. The paper improves upon the traditional CRO to solve our BRP. The results demonstrate that the enhanced version performs better than the traditional one in terms of solution quality.

4. This paper proposes novel operators and concepts of neighbor-node sets that can also be incorporated into other heuristics to solve routing problems, including but not limited to BRPs.

The reminder of this paper is as follows. Section 'Formulation' depicts the problem formulation. Section 'Solution method' describes the solution method. Section 'Numerical studies' presents the numerical results. Finally, Section 'Conclusion' provides conclusions.

Formulation

The bike-sharing system is represented by a complete direct graph $G_0 = (V_0, A_0)$, where V_0 and A_0 are sets of nodes and arcs, respectively. The set of nodes is composed by a set of stations denoted by $V = \{1, 2, \ldots, N\}$ and the depot (denoted by 0), where N is the number of stations. Each node is characterized by its initial inventory s_i^0, final inventory s_i, demand q_i, and capacity C_i (in terms of number of bicycles). If $q_i < s_i^0$ $(i \in V)$, station i is a pickup station, belongs to the set of pickup stations S, and can provide $s_i^0 - q_i$ bicycles to the drop-off stations. If $q_i = s_i^0$ $(i \in V)$, station i is called a balanced station and belongs to the set of balanced stations B. If $q_i > s_i^0$ $(i \in V)$, station i is a drop-off station, belongs to the set of drop-off stations D, and should ideally be supplied by $q_i - s_i^0$ bicycles.

A single vehicle with the capacity Q is used to redistribute bicycles within the repositioning duration T. It starts from the depot empty and travels to assigned stations to load and unload bikes. It may not visit all stations due to time limitations. It picks up y_i^L bikes at an assigned pickup station $(i \in S)$ and drops off y_i^U bikes at a visited drop-off station $(i \in D)$. It visits each station no more than once before finally returning to the depot empty at the end of the operation. The route is formed by a sequence of arcs. Whether an arc is on the route is defined by a binary variable x_{ij} $(i,j \in V_0)$. It equals one if $(i,j) \in A_0$ is on the route; and zero otherwise. Each arc $(i,j) \in A_0$ is associated with an operational time t_{ij} that includes the travel time along the arc and the expected time required to load or unload bicycles at station j (see Rainer-Harbach et al. (2015)). Each arc is also associated with a flow variable f_{ij} $(i,j \in V_0)$, which represents the number of bikes on the vehicle when traveling from node i to node j.

The repositioning problem is formulated as follows.

$$\text{Min } z = \sum_{i \in V} \psi_i + \mu \cdot \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} \tag{1}$$

s.t.

$$\psi_i \geqslant q_i - s_i, \quad \forall i \in V \tag{2}$$

$$\sum_{j \in V} x_{0j} = 1, \tag{3}$$

$$\sum_{j \in V_0, j \neq i} x_{ij} \leqslant 1, \quad \forall i \in V \tag{4}$$

$$\sum_{j \in V_0, j \neq i} x_{ij} = \sum_{j \in V_0, j \neq i} x_{ji}, \quad \forall i \in V_0 \tag{5}$$

$$\sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} \leqslant T, \tag{6}$$

$$g_j \geqslant g_i + 1 - G(1 - x_{ij}), \quad \forall i \in V_0,\ j \in V,\ i \neq j \tag{7}$$

$$\sum_{j \in V} f_{0j} = 0, \tag{8}$$

$$f_{ij} \leqslant Q \cdot x_{ij}, \quad \forall i \in V_0,\ j \in V_0,\ i \neq j \tag{9}$$

$$s_i < C_i, \quad \forall i \in V \tag{10}$$

$$s_i = s_i^0 - y_i^L + y_i^U, \quad \forall i \in V \tag{11}$$

$$y_i^L - y_i^U = \sum_{j \in V_0, j \neq i} f_{ij} - \sum_{j \in V_0, j \neq i} f_{ji}, \quad \forall i \in V \tag{12}$$

$$y_i^U = \min\left(\max(q_i - s_i^0, 0) \cdot \sum_{j \in V_0, j \neq i} x_{ij}, \sum_{j \in V_0, j \neq i} f_{ji}\right), \quad \forall i \in V \tag{13}$$

$$y_i^L \leqslant \min\left(\max(s_i^0 - q_i, 0) \cdot \sum_{j \in V_0, j \neq i} x_{ij}, \left(Q - \sum_{j \in V_0, j \neq i} f_{ji}\right)\right), \quad \forall i \in V \tag{14}$$

$$\sum_{i \in V} y_i^L = \sum_{i \in V} y_i^U, \tag{15}$$

$$y_i^L + y_i^U \geqslant \sum_{j \in V_0, j \neq i} x_{ij}, \quad \forall i \in V \tag{16}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in V_0,\ j \in V_0,\ i \neq j \tag{17}$$
$$x_{ii} = 0, \quad \forall i \in V_0 \tag{18}$$
$$y_i^L \geqslant 0,\ \text{integer}, \quad \forall i \in V \tag{19}$$
$$y_i^U \geqslant 0,\ \text{integer}, \quad \forall i \in V \tag{20}$$
$$f_{ij} \geqslant 0, \quad \forall i \in V_0,\ j \in V_0,\ i \neq j \tag{21}$$
$$g_i \geqslant 0, \quad \forall i \in V_0 \tag{22}$$
$$\psi_i \geqslant 0, \quad \forall i \in V \tag{23}$$

where μ is the weight associated with the vehicle's total operational time and G is the number of nodes involved in the network (i.e., $G = N + 1$). g_i is an auxiliary variable associated with each node to define the sub-tour elimination constraint. ψ_i is an auxiliary variable associated with each station. Its minimum value, denoted by ψ_i^*, is the unmet demand at station i, i.e., $\psi_i^* = \max(q_i - s_i, 0)$. This condition implies that $\psi_i^* = \min \psi_i$ s.t. $\psi_i \geqslant q_i - s_i$ and $\psi_i \geqslant 0$.

Objective (1) aims to minimize the weighted sum of two terms: the total number of the unsatisfied customers (or total unmet demand) reflected by $\sum_{i \in V} \psi_i$ and the vehicle's total operational time $\sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij}$. Note that $\sum_{i \in V} \psi_i$ is equivalent to $\sum_{i \in V} \psi_i^*$ because the objective function is minimized.

Constraint (2) defines one of the two conditions of the unmet demand at each station. (The second one is defined by condition (23).) Constraint (3) states that the vehicle leaves the depot exactly once. Constraint (4) requires that each station can be visited by the vehicle at most once. Constraint (5) is the vehicle flow conservation condition, which ensures that the vehicle must leave the assigned station after visiting that station and that the vehicle must return to the depot. Constraint (6) ensures that the repositioning operation must be finished within the repositioning duration T. Constraint (7) is used to eliminate sub-tours. Constraint (8) ensures that the vehicle leaves the depot empty. Constraint (9) is the vehicle capacity constraint; it ensures that the number of bicycles on the vehicle on each arc is not greater than the vehicle capacity Q. Constraint (10) requires that the final inventory at each station should not exceed the station capacity.

Constraints (11)–(16) are all related to the loading and unloading quantities. Constraint (11) depicts the conservation of bicycles at each station; it defines the final inventory of each station. Constraint (12) states the conservation of bicycles on each vehicle: the difference between loading and unloading quantities at a station equals the difference between the quantities of bicycles on the vehicle when it reaches and leaves the station. Constraint (13) ensures that the unloading quantities equal the minimum value of two terms: the number of additional bicycles needed by station i and the number of bicycles on the vehicle when the vehicle reaches station i. If the station $i \in S$ ($q_i < s_i^0$), $\max(q_i - s_i^0, 0)$ will equal 0, which makes the unloading quantity y_i^U equal 0. Thus, constraint (13) implies that the unloading quantities at all pickup stations are zeros. Constraint (14) requires that the loading quantity at a station should not be greater than the number of bicycles supplied by that station and greater than the available spaces on the vehicle when it arrives at the station. The loading quantity at a station is not simply subject to the station condition itself and the vehicle capacity but also depends on the number of bicycles needed by the drop-off stations along the rest of the vehicle route. For this reason, "$\leqslant$" is used in constraint (14) rather than "=". If the station $i \in D$ ($q_i > s_i^0$), the loading quantity y_i^L is zero, because $\max(s_i^0 - q_i, 0)$ is zero. Therefore, constraint (14) limits the loading quantities y_i^L at drop-off stations to zero. Constraint (15) ensures that all bicycles loaded onto the vehicle are unloaded at the end of the repositioning operation. Constraint (16) implies that the loading/unloading operation must be carried out at each station visited by the vehicle.

Constraint (17) defines x_{ij} as a binary variable. Constraint (18) states that the vehicle cannot travel from a node to the same node. Constraints (19) and (20) ensure that the loading and unloading quantities are non-negative integers. Constraints (21)–(23) state that the number of bicycles on the vehicle and the auxiliary variables are all non-negative.

Solution method

In this section, an enhanced version of CRO is proposed to solve the static repositioning problem. For both the original CRO and the enhanced CRO, the station characteristics are used in the operators and a solution adjustment strategy is considered to ensure the feasibility and quality of the solutions. These heuristics are also mainly used to handle route structures, and a subroutine is proposed and incorporated into them to determine the loading and unloading quantities at each visited station. The major differences between the original CRO and the enhanced CRO are stated as follows. First, newly defined operators are applied to obtain new solutions. Second, two neighbor-node sets are considered to narrow the solution search space. Third, an intensive search method is added to ensure a sufficient local search. Fourth, some of the elementary reactions and rules are adjusted to improve the solution quality.

Chemical reaction optimization

The major components of the CRO framework are molecules and elementary reactions.

Molecules

Molecules, denoted by M, are characterized by the following properties.

(1) Molecular structure ω: It corresponds to a solution to the problem. In the bike repositioning problem, the molecular structure corresponds to a vehicle route.
(2) Potential energy (PE): It quantifies the molecular structure in terms of the objective function value in the optimization problem. i.e., PE corresponds to the objective function value.
(3) Kinetic energy (KE): It measures the tolerance of having a worse solution and represents the ability to escape from a local minimum.

Elementary reactions

Elementary reactions occur as a result of the conservation of energy. They are distinct from one another in their ways of manipulating the energies of reactant molecules. Normally, four types of elementary reactions are taken into consideration, including (1) on-wall ineffective collision, (2) decomposition, (3) inter-molecular ineffective collision, and (4) synthesis.

On-wall ineffective collision. An on-wall ineffective collision refers to a reaction in which a molecule M hits the wall of the container and bounces back. The molecular structure of the resultant molecule M' can be expressed as $\omega' = Onwall(\omega)$, where *Onwall* is a neighborhood search operator that is used to obtain the resultant molecule's structure by modifying the structure of a reactant within a small extent. If the resultant molecule possesses a lower PE than the total energy of a reactant molecule, on-wall ineffective collision will take place and the molecular structure stored in the memory will be updated to ω'. Mathematically, the if-condition can be stated as

$$PE_\omega + KE_\omega \geqslant PE_{\omega'}, \tag{24}$$

where PE_ω and KE_ω are the potential and kinetic energy held by the reactant whose molecular structure is represented by ω. In this type of reaction, a fraction of KE lost to the environment is considered, and this process is controlled by the parameter *KELossRate*, where *KELossRate* is the maximum percentage of KE loss. A random number $q \in [KELossRate, 1]$ determines the fraction of KE that is possessed by the resultant molecule. Based on the theory of energy conservation, the KE of the molecule with its structure ω' and KE loss (denoted by *KEloss*) can be respectively calculated by

$$KE_{\omega'} = (KE_\omega + PE_\omega - PE_{\omega'}) \times q, \quad \text{and} \tag{25}$$

$$KEloss = (KE_\omega + PE_\omega - PE_{\omega'}) \times (1 - q). \tag{26}$$

The energy loss is stored in the central *buffer* to support decomposition, which is depicted next. If inequality (24) does not hold, the on-wall ineffective collision is not permitted to occur and the molecular structure saved in the memory retains ω.

The pseudocode of the on-wall ineffective collision is as follows:

Subroutine 1. *OnwallIneffCollision*(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. Obtain $\omega' = Onwall(\omega)$
2. Calculate $PE_{\omega'}$
3. **if** $PE_\omega + KE_\omega \geqslant PE_{\omega'}$ **then**
4. Get q randomly in interval $[KELossRate, 1]$
5. Calculate KE of resultant molecule
6. Update $buffer = buffer + (KE_\omega + PE_\omega - PE_{\omega'}) \times (1 - q)$
7. Update the profile of M by $\omega = \omega'$, $PE_\omega = PE_{\omega'}$ and $KE_\omega = KE_{\omega'}$
8. **end if**
9. **Output** M and *buffer*

Decomposition. The process in which a molecule hits the wall and decomposes into two molecules is called decomposition. Unlike ineffective collision, decomposition is a violent reaction after which the molecular structures of the two resultant molecules ω'_1 and ω'_2 are very different from the original molecule with structure ω. The two molecular structures are obtained by the *Decom* operator:

$$Decom(\omega) = [\omega'_1, \omega'_2].$$

The molecule decomposes into two molecules if the total potential energy of the resultant molecules is less than the total energy of the reactant:

$$PE_\omega + KE_\omega \geqslant PE_{\omega'_1} + PE_{\omega'_2}. \tag{27}$$

The KE of the two new molecules are

$$KE_{\omega'_1} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2}) \times k, \quad \text{and} \tag{28}$$

$$KE_{\omega'_2} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2}) \times (1-k), \tag{29}$$

where $k \in [0,1]$ is a random number to help assigning KE to the resultants.

Given inequality (27), decomposition only occurs when KE_ω is sufficiently large. The KE of the resultant molecules are calculated according to Eqs. (28) and (29). However, the KE of the molecule continues to decrease in a sequence of on-wall ineffective collisions due to energy loss. If the reactants do not possess sufficient total energy for decomposition, an alternative criterion stated below for decomposition to occur is checked:

$$PE_\omega + KE_\omega + buffer \geqslant PE_{\omega'_1} + PE_{\omega'_2}. \tag{30}$$

If the above holds, the energy saved in the central *buffer* is adopted to support the decomposition and $KE_{\omega'_1}$ and $KE_{\omega'_2}$ are obtained by:

$$KE_{\omega'_1} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2} + buffer) \times m_1 \times m_2, \quad \text{and} \tag{31}$$

$$KE_{\omega'_2} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2} + buffer) \times m_3 \times m_4, \tag{32}$$

where m_1, m_2, m_3, and m_4 are independent random numbers within [0,1].

If decomposition occurs under either of the two conditions (27) and (30), the molecular structures of the resultants are stored in the memory and the original one is removed. If neither condition holds, the reaction cannot occur and the molecular structure retains ω.

The pseudocode of the decomposition is as follows:

Subroutine 2. *Decomposition*(*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. Obtain $[\omega'_1, \omega'_2] = Decom(\omega)$
2. Calculate $PE_{\omega'_1}$ and $PE_{\omega'_2}$
3. Let $temp_1 = PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2}$
4. Create a Boolean variable *Success*
5. **if** $temp_1 \geqslant 0$ **then**
6. *Success* = TRUE
7. Create two new molecules M'_1 and M'_2
8. Get k randomly in interval [0,1]
9. Calculate KE of resultant molecules
10. Calculate KE of resultant molecules
11. 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
12. **else if** $temp_1 + buffer \geqslant 0$ **then**
13. *Success* = TRUE
14. Get m_1, m_2, m_3, and m_4 independently randomly in interval [0,1]
15. Calculate KE of resultant molecules
16. Calculate KE of resultant molecules
17. Update $buffer = temp_1 + buffer - KE_{\omega'_1} - KE_{\omega'_2}$
18. 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
19. **else**
20. *Success* = FALSE
21. **end if**
22. **Output** M'_1, M'_2, *Success*, and *buffer*

Inter-molecular ineffective collision. Two molecules, M_1 and M_2, are involved in inter-molecular ineffective collision. They collide and bounce away. This reaction is not vigorous. The resultants' molecular structures, ω'_1 and ω'_2, are obtained by the *Inter* operator:

$$Inter([\omega_1, \omega_2]) = [\omega'_1, \omega'_2].$$

Inter-molecular ineffective collision occurs if the following energy requirement is satisfied:

$$PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geqslant PE_{\omega'_1} + PE_{\omega'_2}. \tag{33}$$

A random number $p \in [0,1]$ is then used to determine the KE of the resultant molecules, as stated below.

$$KE_{\omega'_1} = (PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'_1} - PE_{\omega'_2}) \times p, \quad \text{and} \tag{34}$$

$$KE_{\omega'_2} = (PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'_1} - PE_{\omega'_2}) \times (1-p). \tag{35}$$

The molecular structures ω_1 and ω_2 saved in the memory are replaced by the newly obtained structures ω_1' and ω_2'. If inequality (33) does not hold, the molecular structures remain unchanged.

The pseudocode of the inter-molecular ineffective collision is as follows:

Subroutine 3. *InterMoleIneffCollision*(M_1, M_2)

Input: Molecule M_1 with its profile and M_2 with its profile
1. Obtain $[\omega_1', \omega_2'] = Inter([\omega_1, \omega_2])$
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. Calculate $temp_2$
4. **if** $temp_2 \geqslant 0$ **then**
5. Get p randomly in interval [0,1]
6. Calculate KE of resultant molecule
7. Calculate KE of resultant molecule
8. 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 $KE_{\omega_2} = KE_{\omega_2'}$
9. **end if**
10. **Output** M_1 and M_2

Synthesis. Synthesis refers to the reaction of two molecules that collide and combine with each other. A new molecular structure ω' is obtained by synthesizing the two molecules with their structures ω_1 and ω_2. The operator is defined as:

$$Synth([\omega_1, \omega_2]) = \omega'.$$

Synthesis occurs if the following condition is met:

$$PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geqslant PE_{\omega'}. \tag{36}$$

The KE of the new molecular structure is given by

$$KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}. \tag{37}$$

$PE_{\omega'}$ is supposed to have a similar value to PE_{ω_1} and PE_{ω_2}. According to Eq. (37), $KE_{\omega'}$ should be much larger than KE_{ω_1} and KE_{ω_2}, which means that the probability of the resultant escaping from a local minimum is increased.

The pseudocode of the synthesis is as follows:

Subroutine 4. *Synthesis*(M_1, M_2)

Input: Molecules M_1 and M_2 with their profiles
1. Obtain $\omega' = Synth([\omega_1, \omega_2])$
2. Calculate $PE_{\omega'}$
3. Create a Boolean variable *Success*
4. **if** $PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geqslant PE_{\omega'}$ **then**
5. *Success* = TRUE
6. Create one molecules M'
7. $KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}$
8. Assign ω', $PE_{\omega'}$ and $KE_{\omega'}$ to the profile of M'
9. **else**
10. *Success* = FALSE
11. **end if**
12. **Output** M' and *Success*

Application to the BRP

Solution representation

In the BRP, it is assumed that the vehicle always starts and finishes the repositioning exercise at the depot; thus the depot must be the first and last element in the routing sequence array for all feasible solutions. Therefore, the depot is not present in the sequence, and a solution is represented by a sequence of selected stations visited by the vehicle as shown in Fig. 1. This sequence is presented by $\omega = (r^1, r^2, \ldots, r^n)$, $r^l \in V$, $l = 1, 2, \ldots, n$ where n is the total number of stations visited by the vehicle. However, for notation purposes, we define $r^0 = r^{n+1} = 0$ (where 0 represents the depot).

Neighbor-node sets

To speed up the evaluation of potentially promising neighbor solutions in the search algorithm, two sorted neighbor-node sets for each node $i \in V_0$ are created a priori.

3	2	7	5	1	6	4	9

Fig. 1. Solution representation.

(1) *NodeNeighborC*$_i$:
The set is created as follows:
- $\{j \in \mathrm{V} \setminus \{i\} | \gamma$ nodes with the largest $|s_j^0 - q_j|\}$
- The nodes in *NodeNeighborC*$_i$ are then sorted in ascending order according to operational time t_{ij}, $j \in$ *NodeNeighborC*$_i$.

(2) *NodeNeighborD*$_i$:
The set is created as follows:
- $\{j \in \mathrm{V} \setminus \{i\} | \gamma$ nodes with the smallest $t_{ij}\}$
- The nodes in *NodeNeighborD*$_i$ are then sorted in ascending order according to $s_j^0 - q_j$, $j \in$ *NodeNeighborD*$_i$.

The value of γ is related to the repositioning duration T and the size of the considered network N as follows.

$$\gamma = \min\left((N - 1 - |\mathrm{B}|), \lfloor (2n_{\max}^T + N/2)/2 \rfloor, 3n_{\max}^T\right). \tag{38}$$

This equation is defined on the basis of the preliminary results of the CRO. $n_{\max}^T$ is the estimated maximum number of stations that could be visited by the vehicle during the repositioning duration T. For each instance, $n_{\max}^T$ is obtained by running the CRO once. The value of γ is the minimum of the following three terms: (1) the total number of stations excluding the considered station i and the balanced stations; (2) the average of $2n_{\max}^T$ and $N/2$; and (3) $3n_{\max}^T$. In the case in which N is large, consideration of only twice the maximum number of stations possible in a vehicle route may not be adequate to find good solutions. Hence in the second term, the average of $2n_{\max}^T$ and $N/2$ is taken into account to address the problem. Adding the third term to Eq. (38) ensures that an adequate number of stations are considered for small-size instances and to avoid an excessive number of stations to be considered for large-size instances. γ is the number of stations considered for each node in each neighbor-node set, and thus the total number of the stations considered in the sets *NodeNeighborC*$_i$ and *NodeNeighborD*$_i$ is much larger than the value of γ.

Initial solution construction and determination of loading and unloading quantities

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

Step 1: The best three pickup nodes from *NodeNeighborC*$_0$ are considered. One of them is randomly chosen to insert into position 1 of the routing sequence ω. Set $l = 2$.
Step 2: A node is randomly chosen from *NodeNeighborD*$_{r^{l-1}}$ and inserted into position l of the routing sequence. The insertion can only be performed if it does not lead to violation of constraint (6). Set $l = l + 1$. Repeat this step until constraint (6) is violated.
Step 3: Given the routing sequence ω constructed by the sequential heuristic, the number of bikes to load or unload at each node can be calculated according to constraints (13)–(15) as shown in the following steps:
Step 3.1. Set the number of bikes on the vehicle $F_{bike} = 0$. Set $l = 1$. The station at position l is $v = r^l$.
Step 3.2. If $v \in \mathrm{S}$, the numbers of bikes to load and unload at station v are calculated by $y_v^L = \min(s_v^0 - q_v, Q - F_{bike})$ and $y_v^U = 0$; If $v \in \mathrm{D}$, the numbers of bikes to load and unload at station v are $y_v^L = 0$ and $y_v^U = \min(q_v - s_v^0, F_{bike}, C_v - s_v^0)$. Update the number of bikes on the vehicle by $F_{bike} = F_{bike} + y_v^L - y_v^U$ and $l = l + 1$. Repeat this step until $l = n$.
Step 3.3. If $F_{bike} = 0$, stop.
Step 3.4. If $v = r^l \in \mathrm{D}$, $l = l - 1$.
Step 3.5. If $v = r^l \in \mathrm{S}$, the possible change in the loading quantity at station v is calculated: $\Delta = \min(y_v^L, F_{bike})$. y_v^L and F_{bike} are then revised by $y_v^L = y_v^L - \Delta$ and $F_{bike} = F_{bike} - \Delta$. $l = l - 1$. Go to step 3.3.

Solution adjustment

In this section, three operators are introduced to adjust or improve the routes obtained from the elementary reactions. The main ideas of adjustment include (1) adding stations to a route to make full use of the repositioning duration, (2) making infeasible solutions (due to violation of the time constraint) feasible by deleting nodes, and (3) rearranging the order of the visited stations by 2-Opt to improve the solution quality. All of these operators are applied to modify the routes. For the operators introduced in Sections 'Insertion of stations' and 'Removal of stations', the loading/unloading quantities are recalculated according to step 3 of Section 'Initial solution construction and determination of loading and unloading quantities' only if a new route is obtained after the operators are used. For the operator introduced in Section '2-Opt', the loading/unloading quantities are calculated once a new feasible route is obtained.

Insertion of stations. InsertionOfStations() is triggered if $T - \sum_{i,j\in V_0, i\neq j} t_{ij} \cdot x_{ij} > 0.5t'$, where $t' = \sum_{i,j\in V_0, i\neq j} t_{ij}/(|V_0|^2 - |V_0|)$. The following is the detailed procedure:

Step 1: The type of station v must be determined. This is done by comparing the number of bikes that can be picked up (Y_s) from the visited pickup stations with the number of bikes that can be dropped off (Y_d) at the visited drop-off stations. The number of bikes is calculated as follows: $Y_s = \sum_{i\in S\cap\omega} \min(s_i^0 - q_i, Q)$ and $Y_d = \sum_{i\in D\cap\omega} \min(q_i - s_i^0, Q)$. If $Y_s > Y_d$, station v should be a drop-off station; otherwise, it should be a pickup station.
Step 2: Randomly pick a position g in the routing sequence for insertion of station v.
Step 3: Ω is defined as the intersection of two sets $NodeNeighborD_{r^{g-1}}$ and $NodeNeighborD_{r^g}$. If Ω is not empty, then station $v = \arg\max_{i\in\Omega}|s_i^0 - q_i|$. Otherwise, $v = \arg\min_{i\in NodeNeighborC_{r^{g-1}}} t_{r^{g-1}i}$. Ω may only consist of the type of station determined in Step 1.
Step 4: Station v is inserted into position g if the insertion does not violate constraint (6).
Step 5: Steps 1–4 are repeated until it is no longer possible to insert more stations.

Removal of stations. RemovalOfStations() is triggered when the solutions obtained by applying any one of the modified reaction operators introduced in Section 'Reactions in the enhanced CRO' are infeasible due to violation of constraint (6). To determine the type of station to be removed, it is necessary to compare Y_s with Y_d. If $Y_s > Y_d$, a pickup station should be removed; otherwise, a drop-off station should be removed. This continues until the solution becomes feasible.

2-Opt. 2-*Opt*() is applied to a feasible solution ω in the enhanced CRO. This operator was originally developed for the traveling salesman problem (Lin, 1965) to reduce the total distance traveled. This procedure considers the inversion of all possible sub-sequences (that consist of at least two nodes). The best improved *feasible* solution according to the objective function replaces ω, and the procedure stops. This differs from a traditional local search in which the search continues until no improvement can be found. Fig. 2 shows the inversion of a sub-sequence of four nodes.

Reactions in the enhanced CRO

The four reactions in the enhanced CRO make use of reaction operators to generate new routes. The loading/unloading quantities at the stations are calculated as in step 3 in Section 'Initial solution construction and determination of loading and unloading quantities' once a new feasible route is obtained. The four reactions are given in the following subsections.

On-wall ineffective collision. The reaction uses the *NewOnwall* operator, which is a hybrid operator that consists of the replacement operator *repl*(ω) and the swapping operator *swap*(ω). A new routing sequence is obtained by applying this operator to an original routing sequence. In this process, the number of stations visited by the vehicle is unchanged and involves two main steps. First, the stations whose $y_i^L = 0$ and $y_i^U = 0$ are singled out, and the type (pickup or drop-off) of these stations are identified. Second, either the replacement operator or the swapping operator is applied to obtain a new solution where the choice is controlled by $\lambda \in (0, 1)$. If $\lambda > SwapWeight$, where *SwapWeight* is the probability of choosing the swapping operator, the replacement operator is applied to obtain new solutions; otherwise, the swapping operator is used. The details of how these two operators work are stated below.

(1) Replacement
This operator replaces a node $i \in \omega$ with another node $j \notin \omega$. Either of the following cases will occur.
- $\exists y_v^L = 0$ and $\exists y_v^U = 0\ \forall v \in \omega$: The first node in ω whose $y_v^L = 0$ and $y_v^U = 0$ is denoted as i. If $i \in S$ (or $i \in D$), then it is replaced by $j \in D \setminus (D \cap \omega)$ (or $j \in S \setminus (S \cap \omega)$).
- $\nexists y_v^L = 0$ or $\nexists y_v^U = 0\ \forall v \in \omega$: Node $i \in \omega$ is randomly chosen and replaced by $j \notin \omega$, which is also randomly chosen.

If we denote that node i is located in position h in the routing sequence, then the node located in position $h - 1$ is r^{h-1}. The replacement node j is randomly selected from a set of six nodes. This set is composed of the best three nodes of $NodeNeighborC_{r^{h-1}}$ and the best three nodes of $NodeNeighborD_{r^{h-1}}$.

Fig. 3 shows an example of how *repl*(ω) works. An integer in the routing sequence row represents a station number. An integer in the loading instruction row stands for the number of loading/unloading quantities at the station concerned. Here, a

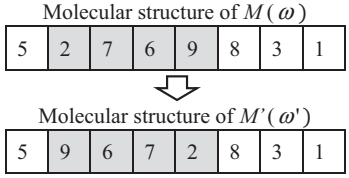

Fig. 2. One possible solution obtained by 2-Opt.

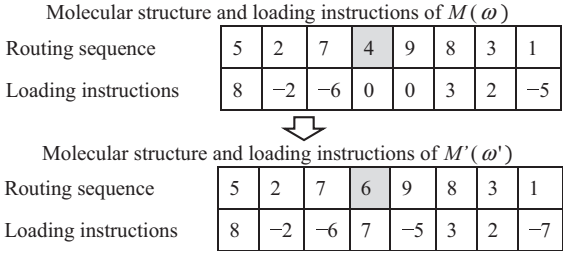

Fig. 3. *repl*(ω) for the on-wall ineffective collision.

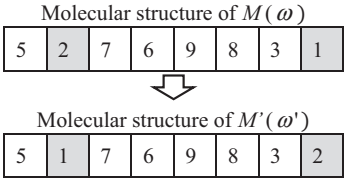

Fig. 4. *swap*(ω) for the on-wall ineffective collision.

positive integer refers to loading quantities and a negative integer refers to unloading quantities. In this example, station 4 is a drop-off station; it is the first station with a zero-loading instruction in the original routing sequence ω. Applying the replacement operator to the original routing sequence ω, station 4 is replaced by a pickup station (station 6), which is chosen from the neighbor-node sets of station 7. Because the new route obtained by the replacement operator is feasible, the loading/unloading quantities are then recalculated. The total number of loading quantities at the pickup stations is 20, which is the same as the total number of unloading quantities at the drop-off stations in the new solution.

(2) Swapping

This operator swaps node $i \in \omega$ with $j \in \omega$. Either of the following cases will occur.

- $\exists y_v^L = 0$ and $\exists y_v^U = 0\ \forall v \in \omega$: The first node in ω whose $y_v^L = 0$ and $y_v^U = 0$ is denoted as i. If $i \in \mathsf{S}$ (or $i \in \mathsf{D}$), then it is swapped with $j \in \omega \cap \mathsf{D}$ (or $j \in \omega \cap \mathsf{S}$).
- $\nexists y_v^L = 0$ or $\nexists y_v^U = 0\ \forall v \in \omega$: Non-adjacent nodes $i \in \omega$ and $j \in \omega$ are randomly chosen and swapped.

In Fig. 4, two drop-off stations (stations 1 and 2) are swapped to obtain a new solution.

The on-wall ineffective collision in the enhanced CRO incorporates the solution adjustment operators introduced in Section 'Solution adjustment' in addition to *NewOnwall*. The pseudocode is given as follows.

Subroutine 5. *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. Obtain $\omega' = NewOnwall(\omega)$
2. Calculate $PE_{\omega'}$
3. **if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} > 0.5t'$
4. $\quad \omega' = InsersionOfStations(\omega')$
5. **else if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} < 0$
6. $\quad \omega' = RemovalOfStations(\omega')$
7. **end if**
8. $\quad$ Obtain $\omega' = 2 - Opt(\omega')$
9. **if** $PE_\omega + KE_\omega \geqslant PE_{\omega'}$ **then**
10. $\quad$ Get q randomly in interval $[KELossRate, 1]$
11. $\quad KE_{\omega'} = (KE_\omega + PE_\omega - PE_{\omega'}) \times q$
12. $\quad$ Update $buffer = buffer + (KE_\omega + PE_\omega - PE_{\omega'}) \times (1 - q)$
13. $\quad$ Update the profile of M by $\omega = \omega'$, $PE_\omega = PE_{\omega'}$ and $KE_\omega = KE_{\omega'}$
14. **end if**
15. **Output** M and *buffer*

Remarks: Lines 3–8 are not executed in the original CRO, and line 1 in the modified CRO uses the proposed *NewOnwall* operator instead of the original *Onwall* operator. The original *Onwall* operator only uses the swapping operator.

Decomposition. The decomposition reaction in the enhanced CRO involves the *NewDecom* operator instead of *Decom*. Unlike the *NewOnwall* operator, *NewDecom* applies to one vehicle route ω to generate two new vehicle routes. The reaction involves the following two steps.

Step 1: Two sub-sequences (sub_1 and sub_2) are obtained from ω. The length (in terms of the number of nodes) of these sub-sequences should be approximately the same.
Step 2: Two solutions (ω_1 and ω_2) are then selected randomly from the solution pool *Pop*. New solutions are obtained by combining sub_1 with ω_1 and sub_2 with ω_2.

In step 2, a new solution ω_1' is obtained as follows:

Step 2.1: Two sets are created: $\Omega_1 = \{i : i \in (sub_1 \cup \omega_1) \cap \mathrm{S}\}$ and $\Omega_2 = \{i : i \in (sub_1 \cup \omega_1) \cap \mathrm{D}\}$.
Step 2.2: To create a new solution, only $\lceil n/2 \rceil$ nodes with the largest $|s_i^0 - q_i|$ ($i \in \Omega_1$) and $\lfloor n/2 \rfloor$ nodes with the largest $|s_i^0 - q_i|$ ($i \in \Omega_2$) are considered. The new solution is initiated with the pickup node $i \in \Omega_1$ that is closest to the depot. The remaining nodes in Ω_1 and Ω_2 are chosen randomly one by one and inserted into ω_1' sequentially.

To create the second solution ω_2', repeat the same procedure but using sub_2 and ω_2 instead. Fig. 5 illustrates how to obtain two new solutions by applying $NewDecom(\omega)$ to one solution.

As in the on-wall ineffective collision of the enhanced CRO, the decomposition reaction also incorporates the solution adjustment operators introduced in Section 'Solution adjustment'. The pseudocode is given as follows.

Subroutine 6. *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. Obtain $[\omega_1', \omega_2'] = NewDecom(\omega)$
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. **if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} > 0.5t'$
4. $\quad \omega_1' = InsersionOfStations(\omega_1')$
5. $\quad \omega_2' = InsersionOfStations(\omega_2')$
6. **else if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} < 0$
7. $\quad \omega_1' = RemovalOfStations(\omega_1')$
8. $\quad \omega_2' = RemovalOfStations(\omega_2')$
9. **end if**
10. Obtain $\omega_1' = 2 - Opt(\omega_1')$ and $\omega_2' = 2 - Opt(\omega_2')$
11. Let $temp_1 = PE_\omega + KE_\omega - PE_{\omega_1'} - PE_{\omega_2'}$
12. Create a Boolean variable *Success*
13. **if** $temp_1 \geqslant 0$ **then**
14. $\quad$ *Success* = TRUE
15. $\quad$ Create two new molecules M_1' and M_2'
16. $\quad$ Get k randomly in interval [0,1]
17. $\quad KE_{\omega_1'} = temp_1 \times k$
18. $\quad KE_{\omega_2'} = temp_1 \times (1 - k)$
19. $\quad$ 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'
20. **else if** $temp_1 + buffer \geqslant 0$ **then**
21. $\quad$ *Success* = TRUE
22. $\quad$ Get m_1 m_2, m_3, and m_4 independently randomly in interval [0,1]
23. $\quad KE_{\omega_1'} = (temp_1 + buffer) \times m_1 \times m_2$
24. $\quad KE_{\omega_2'} = (temp_1 + buffer) \times m_3 \times m_4$
25. $\quad$ Update $buffer = temp_1 + buffer - KE_{\omega_1'} - KE_{\omega_2'}$
26. $\quad$ 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'
27. **else**
28. $\quad$ *Success* = FALSE
29. **end if**
30. **Output** M_1', M_2', *Success*, and *buffer*

Remarks: Lines 3–10 are not executed 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 to obtain new solutions. The circular shift operator is mentioned in Section 'Intensive search'.

Inter-molecular ineffective collision. The inter-molecular ineffective collision adopts $NewInter(\omega)$ to generate two new solutions from two existing solutions ω_1 and ω_2. New solutions are obtained by swapping sub-sequences (sub_1 from ω_1 and sub_2

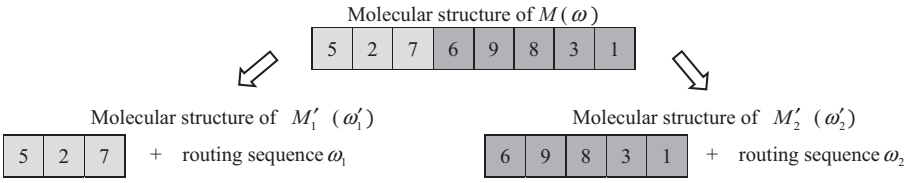

Fig. 5. *NewDecom*(ω) for the decomposition.

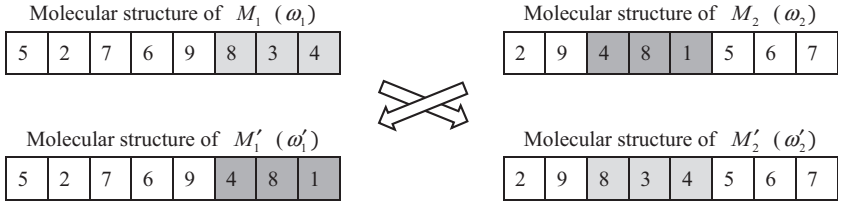

Fig. 6. *NewInter*(ω) for the inter-molecular ineffective collision.

from ω_2) between the existing solutions. The starting position h is first determined randomly from $1,\ldots,n'$, where $n' = \min(n_1, n_2)$ and n_1 and n_2 are the numbers of stations considered by the two solutions. Next, the length of sub_1, L, determined randomly from $1,\ldots,n'-h+1$. The length of sub_2 is then set to be L. The feasible starting positions of sub_2 in ω_2 are therefore $1,\ldots,n_2-L+1$. Some of these feasible starting positions are removed based on the type of the first node of sub_1. Denote i and j as the first node of sub_1 and sub_2, respectively. If $y_i^L = 0$ and $y_i^U = 0$, then the positions with nodes that have the same type as i will be removed because j should be chosen with a different type of station than i. Otherwise, the positions with nodes that have different types from i will be removed because j with the same type as i should be chosen. The remaining feasible positions are then randomly chosen. After exchanging the two subsequences, if $sub_1 \cap (\omega_2 \setminus sub_2) \neq \varnothing$, then the duplicated nodes in sub_1 are replaced with the node found by the replacement operator introduced in Section 'On-wall ineffective collision'. A similar action is performed if $sub_2 \cap (\omega_1 \setminus sub_1) \neq \varnothing$. Fig. 6 shows an example of swapping two subsequences when there are no stations with zero-loading instructions.

Other than *NewInter*(ω), the inter-molecular ineffective collision in the enhanced CRO also incorporates the solution adjustment operators introduced in Section 'Solution adjustment'. The pseudocode is stated below:

Subroutine 7. *NewInterMoleIneffCollision*(M_1, M_2)

Input: Molecule M_1 with its profile and M_2 with its profile
1. Obtain $[\omega_1', \omega_2'] = NewInter([\omega_1, \omega_2])$
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. **if** $T - \sum_{i,j\in V_0, i\neq j} t_{ij}\cdot x_{ij} > 0.5t'$
4. $\quad\omega_1' = InsersionOfStations(\omega_1')$
5. $\quad\omega_2' = InsersionOfStations(\omega_2')$
6. **else if** $T - \sum_{i,j\in V_0, i\neq j} t_{ij}\cdot x_{ij} < 0$
7. $\quad\omega_1' = RemovalOfStations(\omega_1')$
8. $\quad\omega_2' = RemovalOfStations(\omega_2')$
9. **end if**
10. Obtain $\omega_1' = 2-Opt(\omega_1')$ and $\omega_2' = 2-Opt(\omega_2')$
11. Let $temp_2 = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega_1'} - PE_{\omega_2'}$
12. **if** $temp_2 \geqslant 0$ **then**
13. $\quad$Get p randomly in interval $[0,1]$
14. $\quad KE_{\omega_1'} = temp_2 \times p$
15. $\quad KE_{\omega_2'} = temp_2 \times (1-p)$
16. $\quad$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 $KE_{\omega_2} = KE_{\omega_2'}$
17. **end if**
18. **Output** M_1 and M_2

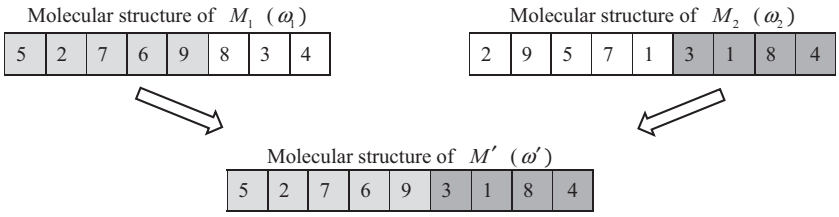

Fig. 7. *Synth*(ω) for the synthesis.

Remarks: Lines 3–10 are not executed in the original CRO, and line 1 in the modified CRO uses the proposed *NewInter* operator instead of the original *Inter* operator. The original *Inter* operator uses the swapping operator, which is the same as the original *Onwall* operator.

Synthesis. Synthesis uses *Synth*(ω) to generate a new solution ω' by combining two existing solutions ω_1 and ω_2, as shown in Fig. 7. Each existing solution is divided into two sub-sequences. The length of each subsequence is randomly determined. The new solution is obtained by combining the first sub-sequence from ω_1 with the second sub-sequence from ω_2. If any of the nodes in ω' are duplicated, then those in the second sub-sequence are replaced by nodes randomly selected from either $NodeNeighborC_i$ or $NodeNeighborD_i$.

The pseudocode of the synthesis, which also incorporates the solution adjustment operators introduced in Section 'Solution adjustment', is presented below.

Subroutine 8. *NewSynthesis*(M_1, M_2)

Input: Molecules M_1 and M_2 with their profiles
1. Obtain $\omega' = Synth([\omega_1, \omega_2])$
2. Calculate $PE_{\omega'}$
3. **if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} > 0.5t'$
4. $\omega' = InsersionOfStations(\omega')$
5. **else if** $T - \sum_{i,j \in V_0, i \neq j} t_{ij} \cdot x_{ij} < 0$
6. $\omega' = RemovalOfStations(\omega')$
7. **end if**
8. Obtain $\omega' = 2 - Opt(\omega')$
9. Create a Boolean variable *Success*
10. **if** $PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geqslant PE_{\omega'}$ **then**
11. *Success* = TRUE
12. Create one molecules M'
13. $KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}$
14. Assign ω', $PE_{\omega'}$, and $KE_{\omega'}$ to the profile of M'
15. **else**
16. *Success* = FALSE
17. **end if**
18. **Output** M' and *Success*

Remarks: Lines 3–8 are not executed in the original CRO, and line 1 in the modified CRO uses the proposed *Synth* operator, which is the same as the one in the original CRO.

Intensive search

Intensive search is composed by two operators: consecutive replacement and circular shift. The operator (either consecutive replacement or circular shift) that is adopted to obtain new solutions in the intensive search is randomly determined. More than one new solution can be generated by the intensive search, but only a *feasible* solution with the minimum objective value is selected and treated as the resultant solution obtained from the intensive search. The energy criterion (i.e., inequality (24)) for the occurrence of on-wall ineffective collision and the energy assignment Eqs. (25) and (26) introduced in Section 'On-wall ineffective collision' are used in the intensive search because the intensive search performs like an on-wall ineffective collision (without changing the total number of solutions in the solution pool after the reaction). **Subroutine 9** shows the framework of the intensive search. *Success* means that the energy criterion (inequality (24)) is satisfied and that the new solution can be included in the solution pool *Pop*. The following are the details of the two operators.

(1) Consecutive replacement

This operator is similar to the replacement operator in Section 'On-wall ineffective collision'. One difference is that it now begins to replace the nodes at the end of the route. Another difference is that instead of replacing just one node, as in the replacement operator, up to five adjacent nodes are replaced.

Step 1: The node located in the last position n of route ω is replaced by an unvisited node chosen from $NodeNeighborC_{r^{n-1}}$ and $NodeNeighborD_{r^{n-1}}$. The replacement is chosen by following the procedure explained in Section 'On-wall ineffective collision' (1). This is the first new solution. Set $\kappa = 1$.
Step 2: Set $l = 0$. The node located in position $h = n - \kappa + l$ is replaced by randomly choosing a node from $NodeNeighborC_{r^{h-1}}$ and $NodeNeighborD_{r^{h-1}}$. Set $l = l + 1$. The replacement continues until $l > \kappa$. This is considered the $(\kappa + 1)$-th new solution.
Step 3: Set $\kappa = \kappa + 1$, and repeat step 2 until $\kappa > \min\{4, n - 1\}$.

With this operator, up to five new solutions are created. The best feasible solution, denoted as $\omega' = Consecutivereplacement(\omega)$, is kept.

(2) Circular shift

This operator generates neighbor solutions by shifting the nodes $r^h, \ldots, r^n$ to the left so that a different pickup station (located in position h) is in position 1 and the stations originally located in positions $1, \ldots, h-1$ are moved to positions $n - h + 2, \ldots, n$, respectively. A maximum of two new solutions are generated with this operator by choosing the two pickup locations (located in position $h \neq 1$) in the route that have the shortest operational time to the depot. The best feasible solution expressed by $\omega' = Circularshift(\omega)$ is retained. Fig. 8 shows that station 6 (a pickup station) is selected and $6, 9, \ldots, 1$ are shifted to the left so that 6 is in position 1.

Subroutine 9. *IntensiveSearch*()

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 t in the interval $(0, 1)$
2. **if** $t > 0.5$ **then**
3. Obtain $\omega' = Consecutivereplacement(\omega)$
4. **else**
5. Obtain $\omega' = Circularshift(\omega)$
6. **end if**
7. Calculate $PE_{\omega'}$
8. **if** *Success* **then**
9. Get q randomly in interval $[KELossRate, 1]$
10. Calculate $KE_{\omega'}$ by Eq. (25)
11. Update *buffer* based on (26) i.e., $buffer = buffer + (KE_\omega + PE_\omega - PE_{\omega'}) \times (1 - q)$
12. Update the profile of M by $\omega = \omega'$, $PE_\omega = PE_{\omega'}$, and $KE_\omega = KE_{\omega'}$
13. **end if**
14. Check for any new minimum solution
15. **Output** M and *buffer*

Implementation of enhanced and original CRO

The implementation details of the original CRO and the enhanced CRO are given in **Algorithm 1** and **Algorithm 2**. The flow charts for the original CRO and the enhanced CRO are shown in Figs. 9 and 10. Similar to the original CRO, the enhanced CRO has three stages, including initialization, iteration, and final stage.

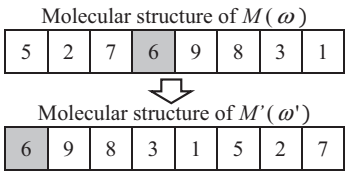

Fig. 8. One possible solution obtained by circular shift.

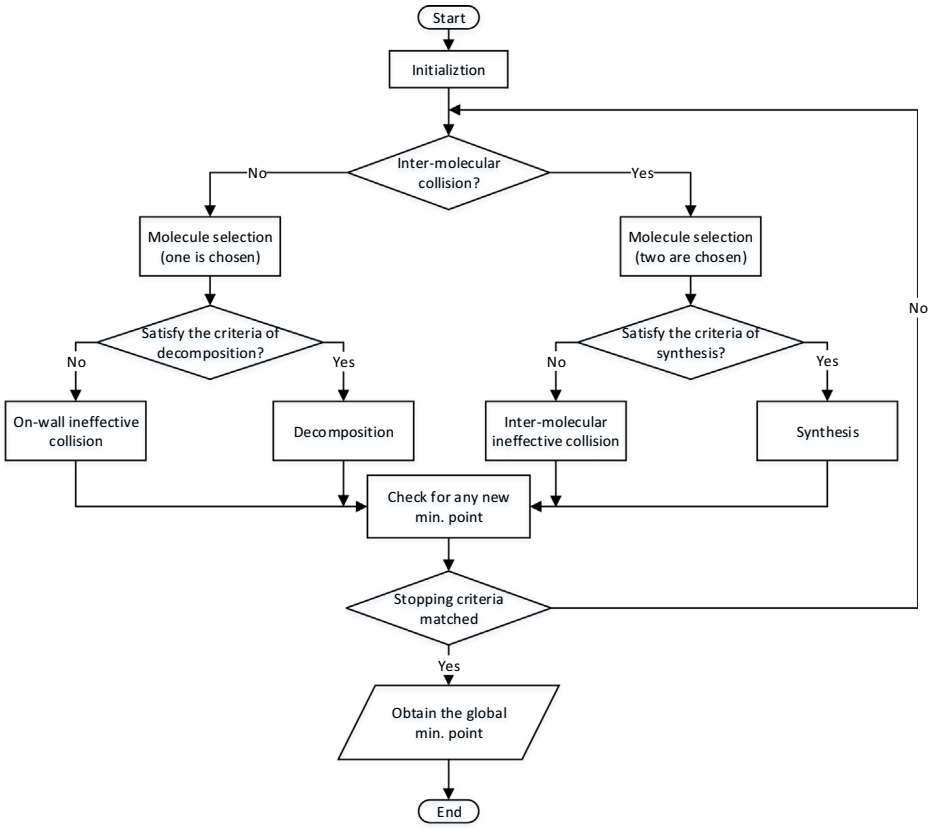

Fig. 9. Flow chart for the original CRO (Lam and Li, 2010).

In the initiation stage (**Subroutine 10**), the values are assigned to the parameters and the initial solutions are generated according to the simple construction heuristic described in Section 'Initial solution construction and determination of loading and unloading quantities'. The current best solution is set to be the feasible solution with the lowest objective function value among all of the initial feasible solutions.

Subroutine 10. *Initialization*()

Input: Problem-specific information (objective 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 objective function value $z(\omega)$.
5. Assign KE_ω with the value of *InitialKE*.
6. Assign $Hit_\omega = 0$
7. **end for**
8. Let the central energy buffer be *buffer* and *buffer* = 0
9. Determine the current best solution among all the initial solutions
10. **Output** *Pop*

The second stage includes the enhanced CRO main steps. Within the main steps, as in the original CRO, there are four principle reactions in the enhanced version. For the enhanced CRO, a hybrid operator ($repl(\omega)$ and $swap(\omega)$) is adopted in the on-wall ineffective collision, whereas only the swapping operator is used in the original CRO. Unlike $NewDecom(\omega)$ in the enhanced CRO, a circular shift is applied to obtain new solutions from the decomposition in the original CRO. For the inter-molecular ineffective collision, the swapping of sub-sequences between solutions is used in the enhanced CRO,

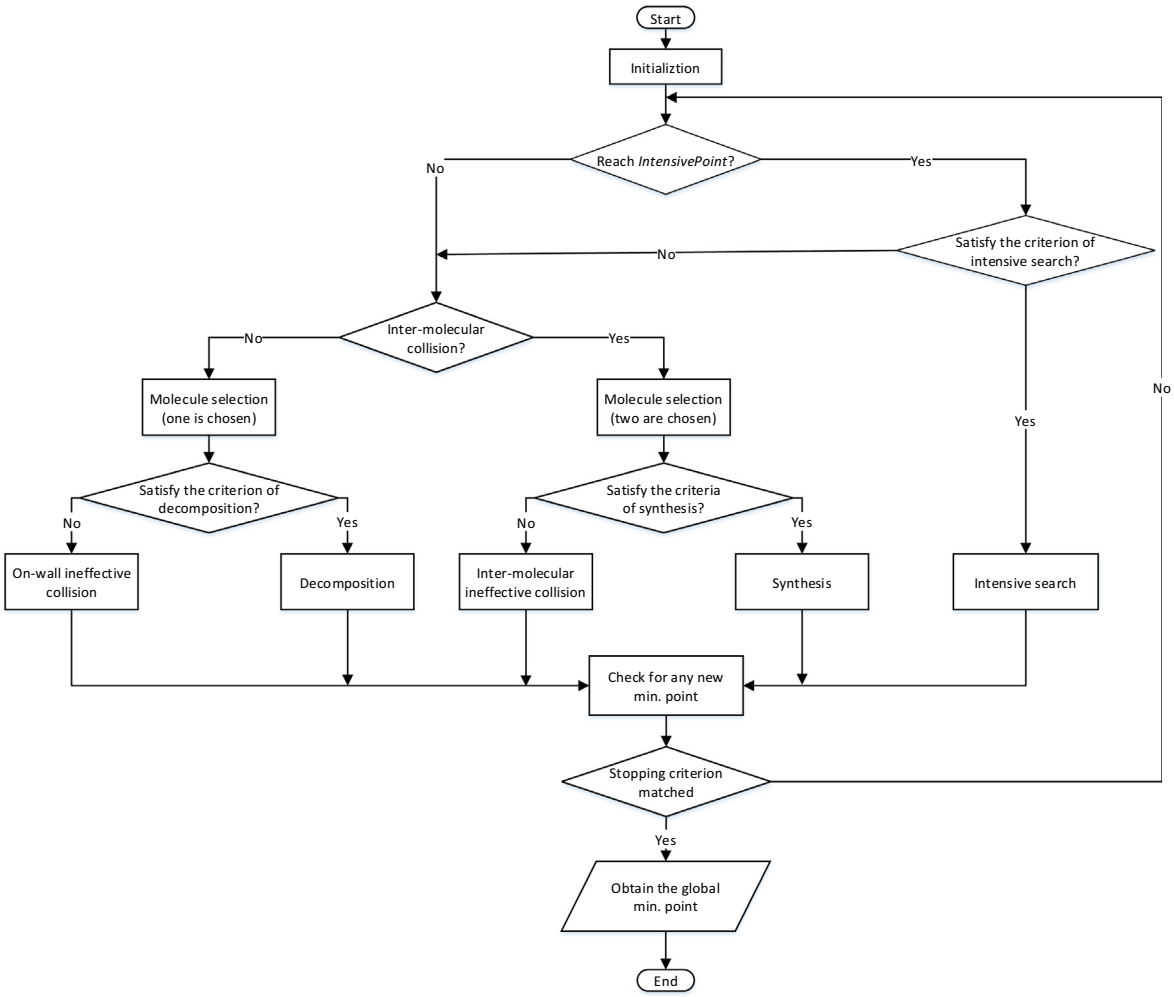

Fig. 10. Flow chart for the enhanced CRO.

whereas the swapping operation, which is used in the on-wall ineffective collision, is used to obtain new solutions in the original CRO. For synthesis, the operators adopted in the enhanced CRO and the original CRO are the same. At the end of each iteration of the enhanced CRO, the current best solution is updated (i.e., the routing sequence and the objective function value is stored in the memory) if a new solution with a lower objective 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 structures of a molecule (i.e., all of the solutions generated from a specific solution).

Another two differences between the original CRO and the enhanced CRO in the main steps are the definition of Hit, Hit_ω, of each molecule and the decomposition criterion. In the enhanced CRO, Hit_ω in the decomposition criterion (**Subroutine 11**, line 4) refers to the total number of consecutive iterations for either the on-wall ineffective collision or the inter-molecular ineffective reactions (i.e., the total number of searches among the neighbor solutions) before the current best solution is updated. For each molecule, if the current best solution is not updated after either the on-wall ineffective collision or the inter-molecular ineffective collision occurs, Hit_ω is increased by one; otherwise, Hit_ω is reset to zero. α in the decomposition criterion (**Subroutine 11**, line 4) is a parameter that represents the maximum number of consecutive iterations to undergo either the on-wall ineffective collision or the inter-molecular ineffective reaction. The decomposition criterion $Hit_\omega - \alpha > 0$ means that no better solution is obtained after searching the neighbor solutions of the solution ω with very similar structures α times. If such a criterion is met, the decomposition will be carried out to obtain a solution with a very different structure. Hit_ω is set to zero after the decomposition or synthesis has occurred. In the original CRO, the decomposition criterion (**Algorithm 1**, line 7) is ($Hit'_\omega - minHit_\omega > \alpha$). Hit'_ω is the number of collisions of any kind. $minHit_\omega$ records the number of

collisions when the solution obtains its current minimum value. Both Hit'_{ω} and $minHit_{\omega}$ are set to 0 when a new molecule is created. It is possible that for a very poor current solution, even though the current best solution continues to be updated, the quality would still be poor compared with other solutions after many reactions have been conducted, the search cannot jump to another solution with a very different structure, and the search is conducted within the local region for a long time. This is different from the enhanced CRO. If a considered solution is far from the current best solution, after searching the neighbor region α times, it will be discarded by the enhanced CRO.

The third difference is the rule for the occurrence of synthesis. In the enhanced CRO, for the decomposition reaction to occur, the number of solutions in *Pop* should be at least three (whereas other reactions require a smaller number). Moreover, the number of solutions in the population is reduced *only* after synthesis occurs. Therefore, in the enhanced CRO, the condition (*PopSize* > 3) is added to the synthesis criterion (**Subroutine 11**, line 15) to avoid *Pop* being too small to allow any reaction, including decomposition, in the subsequent iteration.

The enhanced CRO main steps (see **Subroutine 11**) are run in the first part of the second stage until the total number of iterations reaches *IntensivePoint*. In the second part of the second stage, either intensive search or main enhanced CRO steps is selected, and the selection is controlled by δ. δ is a function of the current number of iterations stated as

$$\delta = 1 - (maxIteration - iter)/(maxIteration - IntensivePoint). \tag{39}$$

If a random number is less than δ, one solution is selected from the solution pool and modified to obtain a new solution with the intensive search; otherwise, the main CRO steps are applied. Based on definition (39) and line 8 of **Algorithm 2** (the criterion of intensive search), the probability of running an intensive search equals δ and varies from 0 to 1 in the second part of the second stage. It increases as the iteration continues.

The second stage repeats until the stopping criterion is matched. The stopping criterion is met if the number of iterations equals the maximum number of iterations *maxIteration*. In the final stage, the best solution obtained is output.

Subroutine 11. *EnhancedCROMainSteps*()

Input: Pool of solutions *Pop*

1. Get random t in the interval $(0, 1)$
2. **if** $t > MoleColl$ **then**
3. Select solution M from *Pop* randomly
4. **if** $Hit_{\omega} - \alpha > 0$ **then**
5. $(M'_1, M'_2, Success) = NewDecomposition(M, buffer)$
6. **if** *Success* = TRUE **then**
7. Remove M from *Pop*
8. Add M'_1 and M'_2 to *Pop*
9. $Hit_{\omega'_1} = 0$ and $Hit_{\omega'_2} = 0$
10. **end if**
11. **else**
12. $NewOnwallIneffCollision(M, buffer)$
13. **else**
14. Select two solutions M_1 and M_2 from *Pop* randomly
15. **if** $KE_{\omega_1} < \beta$, $KE_{\omega_2} < \beta$, and *PopSize* > 3 **then**
16. $(M', Success) = NewSynthesis(M_1, M_2)$
17. **if** *Success* = TRUE **then**
18. Remove M_1 and M_2 from *Pop*
19. Add M' to *Pop*
20. $Hit_{\omega'} = 0$
21. **end if**
22. **else**
23. $NewInterMoleIneffCollision(M_1, M_2)$
24. **end if**
25. **end if**
26. Check for any new minimum solution
27. **Output** *Pop*

Algorithm 1. *Original CRO.*

Pop = *Initialization*()
Let *iter* = 0
while *iter* < *maxIteration* **do**
  Get random t in the interval $(0, 1)$
  if $t > MoleColl$ **then**
    Select solution M from *Pop* randomly
    if $Hit'_{\omega} - minHit_{\omega} > \alpha$ **then**
      $(M'_1, M'_2, Success) = Decomposition(M, buffer)$
    if *Success* = TRUE **then**
      Remove M from *Pop*
      Add M'_1 and M'_2 to *Pop*
      $Hit'_{\omega'_1} = 0$, $Hit'_{\omega'_2} = 0$, $minHit_{\omega'_1} = 0$, and $minHit_{\omega'_2} = 0$
    end if
  else
    $OnwallIneffCollision(M, buffer)$
else
  Select two solutions M_1 and M_2 from *Pop* randomly
  if $KE_{\omega_1} < \beta$ and $KE_{\omega_2} < \beta$ **then**
    $(M', Success) = Synthesis(M_1, M_2)$
    if *Success* = TRUE **then**
      Remove M_1 and M_2 from *Pop*
      Add M' to *Pop*
      $Hit'_{\omega'} = 0$
    end if
  else
    $InterMoleIneffCollision(M_1, M_2)$
  end if
end if
  $iter = iter + 1$
end while
Output the overall best solution and its objective function value

Algorithm 2. *Enhanced CRO.*

Pop = *Initialization*()
Let *iter* = 0
while the stopping criterion is not met **do**
  if *iter* < *IntensivePoint* **then**
    EnhancedCROMainSteps()
  else
    Get random t in the interval $(0, 1)$
    if $t < \delta$ **then**
    IntensiveSearch()
  else
    EnhancedCROMainSteps()
  end if
  end if
    $iter = iter + 1$
end while
Output the overall best solution and its objective function value

Numerical studies

The proposed heuristic was tested on the same set of instances used by Rainer-Harbach et al. (2015), which were generated on the basis of 2011 real-world data provided by Citybike Vienna. The operational time t_{ij} is the average driving time

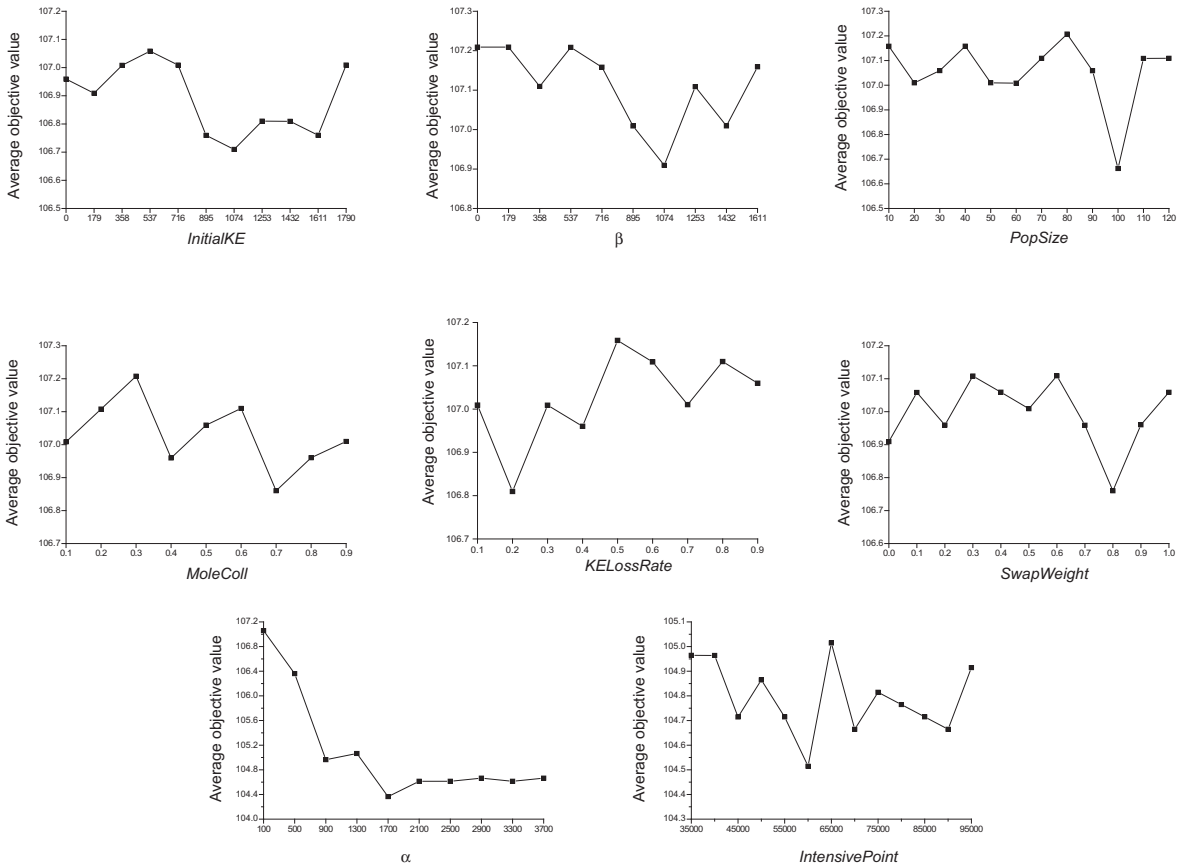

Fig. 11. Parameter tuning for the enhanced CRO.

from node i to j; it considers an estimated time for parking the vehicle and loading and unloading bicycles at node i. The demand at each station was set to be 50% of the station's capacity. In this section, unless otherwise specified, $\mu = 0.00001$, which implies that the number of unsatisfied customers is very important to consider.

The proposed heuristic was coded in Visual Studio using C++, and all computational experiments were carried out on a computer with an Intel i7-3770 CPU @ 3.4 GHz and 32 GB RAM. Unless otherwise specified, "enhanced CRO" refers to the version in which the two neighbor sets are incorporated.

Parameter tuning

The values of the parameters are crucial to the average performance of the proposed heuristic. Therefore, the parameter values are tuned. According to the mechanisms for obtaining new solutions and the criteria of reaction occurrence, the values of parameters related to energy, i.e., *InitialKE* and β, control the ability to obtain new solutions and have a direct and close relationship with the PE of the problem, which usually varies widely among different problems. Therefore, fixing parameter values may not exert their best effects on optimization problems with very different objective function values (Szeto et al., 2014). To tune these parameter values, the method proposed by Szeto et al. (2014) was used. An initial objective value, denoted *InitialObj*, was calculated by $\sum_{i \in V}(\max(q_i - s_i^0, 0))$ to represent the total number of unsatisfied customers before repositioning occurs. *InitialKE* and β were first tuned on the basis of this initial objective value. The trial values used to tune the ratios of these two parameters were the *multiple* of the initial objective value *InitialObj*. Other parameters were tuned in the following order: *PopSize*, *MoleColl*, *KELossRate*, *SwapWeight*, α, and *IntensivePoint*. Twenty random seeds were used. The stopping criterion was set to reach a maximum of 100,000 iterations. The average of the best objective values from 20 runs determined the parameter values.

The test instance used for parameter tuning is a small-size instance consisting of 60 stations. The vehicle capacity is set at 20. The repositioning duration is 21,600 s (6 h). The initial objective value (*InitialObj*) is 179. The initial values of the parameters were set as follows: *InitialKE* = 179; β = 179; *PopSize* = 30; *MoleColl* = 0.5; *KELossRate* = 0.5; *SwapWeight* = 0.5; α = 100; and *IntensivePoint* = 70,000. The first parameter to be tuned was *InitialKE*. This parameter can take on different values from the set $\{0, 179, 358, \ldots, 1790\}$, whereas the other parameters took on their initial values as depicted above. Setting *InitialKE* to 1074 yields the best average results. The next parameter in the tuning process is β, with an *InitialKE* of 1074 and the other

parameters at their initial values, $\beta \in \{179, 360, \ldots, 1611\}$. According to the second plot of Fig. 11, setting β to 1074 yields the best result. The tuning process continues in this manner until every parameter is tuned.

The tuning results are plotted in Fig. 11. The values of the parameters are set as follows: *InitialKE* = 1074 (i.e., 6 × *InitialObj*); β = 1074; *PopSize* = 100; *MoleColl* = 0.7; *KELossRate* = 0.2; *SwapWeight* = 0.8; α = 1700; and *IntensivePoint* = 60,000. These parameter values are applied in the following numerical experiments, except for *InitialKE* and β, for which a ratio of 6 is used. Regarding the value of *MoleColl*, approximately 70% of the overall reactions are bimolecular reactions, which indicates that the operators for the synthesis and inter-molecular ineffective collision reaction are more efficient than those for the decomposition and on-wall ineffective collision reactions for searching for good solutions. The value of *SwapWeight* is far greater than the midpoint 0.5, which shows that the swapping operator performs much better than the replacement operator. However, the average performance of the heuristic for *SwapWeight* values of 0.9 and 1.0 is worse than that for a *SwapWeight* of 0.8, which indicates that the replacement operator is still necessary to obtain good solutions.

Effectiveness of incorporating neighbor-node sets into enhanced CRO

In this section, the Vienna instances studied by Rainer-Harbach et al. (2015) were used. Small-size to medium-size instances are defined as those with 30–90 stations. Large-size instances considered contain 120, 240, and 300 stations. Three repositioning durations were used: T = 14,400 s, T = 21,600 s, and T = 28,800 s. Two vehicle capacities (in terms of number of bicycles) were considered: Q = 10 and Q = 20.

In our study, two neighbor-node sets, *NodeNeighborC*$_i$ and *NodeNeighborD*$_i$, are introduced to improve the efficiency of the proposed heuristic by narrowing the search space. To illustrate the effectiveness of introducing the neighbor-node sets, the comparisons of the performance of the enhanced CRO with and without neighbor-node sets were conducted using the various instances, repositioning durations, and vehicle capacities stated above. To ensure a fair comparison, for each scenario (i.e., a given combination of instance, repositioning duration, and vehicle capacity) and each seed, the enhanced CRO without neighbor-node sets was first executed with a *maxIteration* value of 100,000. The number of solutions generated by the enhanced CRO without neighbor-node sets was then recorded and set as the stopping criterion for the enhanced CRO when incorporating the neighbor-node set(s).

Tables 1–3 present the results obtained with the enhanced CRO without neighbor-node sets and those obtained with the enhanced CRO with individual consideration of *NodeNeighborC*$_i$ and *NodeNeighborD*$_i$ under different repositioning durations. Gap (in percent) in the tables refers to the difference in the average of the best objective values of the concerned scenario obtained from 20 runs by the two variants under comparison, normalized by the average objective value obtained from the enhanced CRO without neighbor-node sets. The value of Gap implies the relative improvement (or deterioration) on the demand satisfaction of the enhanced CRO with neighbor-node sets to the enhanced CRO without the neighbor-node sets. If it is negative, the performance of the enhanced CRO is improved by the introduction of the neighbor-node set; it is weakened otherwise. CPU denotes the average running time in seconds.

As shown in Tables 1–3, almost all of the values for Gap are non-positive (with only three exceptions for the results obtained with the enhanced CRO with *NodeNeighborC*$_i$ and three exceptions for the enhanced CRO with *NodeNeighborD*$_i$), whereas the average running times of the two versions are about the same, which indicates that the general performance of the enhanced CRO with a neighbor-node set is better than that without. It is beneficial to incorporate neighbor-node sets into the CRO to solve the BRP. Moreover, the average standard deviation (SD) decreases when either of the neighbor-node sets is applied to the enhanced CRO, which means that the objective values obtained with the variants are less dispersed and that the variants give more or less similar objective values even if different random seeds are used. We take the results under the repositioning duration T = 28,800 s as an example. The average SD for the results from the enhanced CRO without neighbor-node sets is 1.26 and is greater than that from the enhanced CRO with *NodeNeighborC*$_i$ (0.74) and that with *NodeNeighborD*$_i$ (0.94).

For the results under different repositioning durations (i.e., T = 14,400 s, T = 21,600 s, and T = 28,800 s), the average Gap obtained with the enhanced CRO with *NodeNeighborC*$_i$ have larger absolute values than those obtained with the enhanced CRO with *NodeNeighborD*$_i$, which implies that the overall solution quality of the enhanced CRO with *NodeNeighborC*$_i$ is better for the scenarios considered. However, for some of the scenarios, the Gap obtained with the enhanced CRO with *NodeNeighborC*$_i$ is less negative, which means that the solution quality of the enhanced CRO with *NodeNeighborD*$_i$ is better for the scenario, especially for the scenarios for which the Gap obtained with the enhanced CRO with *NodeNeighborC*$_i$ is positive. For example, regarding the result of T = 14,400 s, $|V| = 90$, and Q = 10, the Gap obtained with the enhanced CRO with *NodeNeighborC*$_i$ is 0.36%, whereas the Gap obtained with the enhanced CRO with *NodeNeighborD*$_i$ is −2.80%. This mixture of conclusions suggests that the enhanced CRO proposed in this paper should consider both the neighbor-node sets *NodeNeighborC*$_i$ and *NodeNeighborD*$_i$ to obtain new solutions for different-size instances, vehicle capacities, and repositioning durations.

Tables 4–6 show a comparison of the results between the enhanced CRO with and without the two neighbor-node sets under different repositioning durations. The enhanced CRO was executed with the same number of solutions generated by the enhanced CRO without neighbor-node sets used as the stopping criterion. The results show that the solution quality of the enhanced CRO is improved by incorporating both neighbor-node sets. By comparing the results with those in Tables 1–3, we found that the solution quality of the enhanced CRO with both neighbor-node sets has been further improved compared to the enhanced CRO without the sets. For example, the average Gap obtained with the enhanced CRO for T = 28,800 s is

Table 1
Result comparison of the enhanced CRO with and without neighbor-node sets (T = 14,400 s).

|V|	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with $NodeNeighborC_i$				Gap (%)	Enhanced CRO with $NodeNeighborD_i$				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD		Best	Average	CPU	SD	
30	10	68.13860	68.13860	1.69	0.00	68.13860	68.13860	1.69	0.00	0.00	68.14040	68.14040	1.68	0.00	0.00
	20	60.14100	60.14181	1.71	0.00	60.14100	60.14118	1.71	0.00	0.00	63.13860	63.13860	1.70	0.00	4.98
60	10	139.13380	139.13380	1.53	0.00	139.13380	139.13380	1.47	0.00	0.00	139.13440	139.13440	1.46	0.00	0.00
	20	126.14400	126.99226	1.61	0.36	127.14160	127.14160	1.56	0.00	0.12	127.14160	127.54169	1.58	0.49	0.43
90	10	253.14400	260.17693	1.60	2.24	261.12060	261.12537	1.54	0.00	0.36	252.14400	252.89352	1.58	0.77	−2.80
	20	241.14340	243.79109	1.62	1.27	241.14400	241.74319	1.61	0.49	−0.84	243.14100	245.04181	1.60	0.44	0.51
120	10	314.13500	314.13569	1.52	0.00	314.13500	314.13500	1.43	0.00	0.00	314.13680	314.13680	1.47	0.00	0.00
	20	295.14400	297.03959	1.54	0.43	294.14400	294.74322	1.48	1.07	−0.77	294.14400	295.34238	1.50	1.16	−0.57
240	10	744.14040	745.02738	1.47	0.30	745.12960	745.12960	1.36	0.00	0.01	736.14400	736.99112	1.44	0.36	−1.08
	20	725.14340	727.23959	1.51	0.94	724.14340	724.74193	1.41	0.86	−0.34	725.14160	725.64139	1.48	0.50	−0.22
300	10	843.12420	843.12774	1.22	0.00	843.13920	843.13920	1.18	0.00	0.00	837.14340	842.08071	1.18	1.85	−0.12
	20	830.14220	834.93914	1.28	1.47	833.14160	833.14160	1.20	0.00	−0.22	828.14400	829.49136	1.26	0.57	−0.65
Avg				1.52	0.58			1.47	0.20	−0.14			1.49	0.51	0.04

Table 2
Result comparison of the enhanced CRO with and without neighbor-node sets (T = 21,600 s).

|V|	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with *NodeNeighborC*$_i$				Gap (%)	Enhanced CRO with *NodeNeighborD*$_i$				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD		Best	Average	CPU	SD	
30	10	47.21420	48.21291	4.43	0.70	47.2142	47.41474	4.47	0.40	−1.66	47.21420	47.31441	4.46	0.30	−1.86
	20	39.21420	40.06171	4.61	0.36	39.21420	39.56243	4.67	0.47	−1.25	39.21420	39.66150	4.65	0.49	−1.00
60	10	120.20700	120.20871	3.79	0.00	115.21420	115.61453	3.83	0.74	−3.82	115.21420	116.01429	3.77	1.08	−3.49
	20	104.21480	106.61054	3.99	0.86	104.21360	104.36450	4.03	0.48	−2.11	104.21480	105.66378	4.00	0.80	−0.89
90	10	231.21000	232.05946	4.17	0.48	231.20760	231.20811	4.19	0.00	−0.37	231.20760	231.45832	4.21	0.43	−0.26
	20	217.21180	218.35928	4.28	0.72	215.21060	215.41246	4.31	0.40	−1.35	215.21180	216.21168	4.36	0.83	−0.98
120	10	289.21540	291.36135	3.92	1.19	287.21240	287.21303	3.96	0.00	−1.42	287.21240	287.41348	3.92	0.40	−1.35
	20	269.21300	271.31084	4.04	0.94	267.21240	268.21255	4.06	0.63	−1.14	268.21240	268.56234	4.04	0.47	−1.01
240	10	717.20820	719.16180	3.78	1.36	715.20700	715.20784	3.56	0.00	−0.55	715.20880	715.26129	3.84	0.22	−0.54
	20	697.21180	701.21099	3.88	1.87	692.21480	693.10703	3.84	0.30	−1.16	692.21180	693.41240	3.95	0.75	−1.11
300	10	817.21240	822.00172	3.37	1.85	813.20400	813.20505	3.21	0.00	−1.07	813.20340	813.25655	3.39	0.22	−1.06
	20	807.21300	809.66069	3.57	1.24	799.21540	801.86300	3.56	0.91	−0.96	800.21180	801.46210	3.63	0.70	−1.01
Avg				3.98	0.97			3.97	0.36	−1.40			4.02	0.56	−1.22

Table 3
Result comparison of the enhanced CRO with and without neighbor-node sets (T = 28,800 s).

$\|V\|$	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with $NodeNeighborC_i$				Gap (%)	Enhanced CRO with $NodeNeighborD_i$				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD		Best	Average	CPU	SD	
30	10	33.28620	34.68218	8.48	0.58	33.28620	34.18464	8.67	0.54	−1.43	33.28680	34.43260	8.58	0.65	−0.72
	20	22.28740	24.68266	8.97	1.02	22.28620	23.68509	9.14	0.73	−4.04	22.28620	23.83446	9.07	0.92	−3.44
60	10	99.28680	102.13227	7.19	1.15	98.28200	98.28263	7.26	0.00	−3.77	98.28200	98.38320	7.30	0.30	−3.67
	20	87.28200	88.23140	7.63	0.59	84.28500	84.93254	7.75	0.47	−3.74	84.28380	84.93188	7.78	0.57	−3.74
90	10	212.27960	213.07834	8.05	0.60	205.28560	206.43485	8.22	0.73	−3.12	206.28380	206.93470	8.25	0.65	−2.88
	20	193.28680	195.88248	8.36	1.24	188.28140	188.93461	8.55	0.57	−3.55	188.28440	190.08323	8.63	0.75	−2.96
120	10	269.27780	271.03248	7.48	0.94	265.28560	266.88437	7.63	0.80	−1.53	266.28800	267.82984	7.59	0.59	−1.18
	20	247.28560	250.53173	7.85	1.51	242.28620	243.48392	8.02	0.51	−2.81	242.28560	243.73467	8.05	0.80	−2.71
240	10	695.28440	698.17972	7.30	1.13	688.28440	693.47567	7.28	2.78	−0.67	687.28740	690.68479	7.55	1.32	−1.07
	20	673.28260	677.93227	7.58	1.74	660.28320	662.98341	7.66	0.90	−2.21	662.28260	664.73434	7.82	1.63	−1.95
300	10	794.28080	798.13128	6.73	2.59	793.26580	793.26871	6.62	0.00	−0.61	787.28560	790.18413	6.93	1.94	−1.00
	20	780.28620	786.53125	7.24	1.99	772.28140	774.33410	7.28	0.86	−1.55	773.28140	775.13425	7.48	1.11	−1.45
Avg				7.74	1.26			7.84	0.74	−2.42			7.92	0.94	−2.23

Table 4
Result comparison of the enhanced CRO with and without two neighbor-node sets (T = 14,400 s).

|V|	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with both sets				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD	
30	10	68.13860	68.13860	1.69	0.00	68.13860	68.13893	1.66	0.00	0.00
	20	60.14100	60.14181	1.71	0.00	60.14100	60.64088	1.69	0.86	0.83
60	10	139.13380	139.13380	1.53	0.00	139.13380	139.13380	1.42	0.00	0.00
	20	126.14400	126.99226	1.61	0.36	126.14400	126.49316	1.58	0.48	−0.39
90	10	253.14400	260.17693	1.60	2.24	252.14400	252.64373	1.58	0.59	−2.90
	20	241.14340	243.79109	1.62	1.27	241.14280	241.99316	1.61	0.85	−0.74
120	10	314.13500	314.13569	1.52	0.00	314.13500	314.13530	1.43	0.00	0.00
	20	295.14400	297.03959	1.54	0.43	294.14400	294.49361	1.48	0.57	−0.86
240	10	744.14040	745.02738	1.47	0.30	736.14400	737.09073	1.66	0.22	−1.07
	20	725.14340	727.23959	1.51	0.94	723.14280	723.39271	1.49	0.43	−0.53
300	10	843.12420	843.12774	1.22	0.00	837.14340	841.33272	1.17	2.49	−0.21
	20	830.14220	834.93914	1.28	1.47	828.14340	828.59298	1.25	0.50	−0.76
Avg				1.52	0.58			1.50	0.58	−0.55

Table 5
Result comparison of the enhanced CRO with and without two neighbor-node sets (T = 21,600 s).

|V|	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with both sets				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD	
30	10	47.21420	48.21291	4.43	0.70	47.21420	47.31414	4.44	0.30	−1.86
	20	39.21420	40.06171	4.61	0.36	39.21420	39.26405	4.66	0.22	−1.99
60	10	120.20700	120.20871	3.79	0.00	115.21420	115.21432	3.81	0.00	−4.15
	20	104.21480	106.61054	3.99	0.86	104.21480	104.66486	4.02	0.59	−1.83
90	10	231.21000	232.05946	4.17	0.48	231.20760	231.20778	4.21	0.00	−0.37
	20	217.21180	218.35928	4.28	0.72	215.21180	215.51297	4.36	0.46	−1.30
120	10	289.21540	291.36135	3.92	1.19	287.21240	287.21267	3.94	0.00	−1.42
	20	269.21300	271.31084	4.04	0.94	267.21540	268.36234	4.08	0.47	−1.09
240	10	717.20820	719.16180	3.78	1.36	715.20700	715.20859	3.77	0.00	−0.55
	20	697.21180	701.21099	3.88	1.87	690.21360	692.26303	3.93	0.86	−1.28
300	10	817.21240	822.00172	3.37	1.85	813.20220	813.20502	3.31	0.00	−1.07
	20	807.21300	809.66069	3.57	1.24	800.21240	800.81228	3.60	0.49	−1.09
Avg				3.98	0.97			4.01	0.28	−1.50

Table 6
Result comparison of the enhanced CRO with and without two neighbor-node sets (T = 28,800 s).

|V|	Q	Enhanced CRO without neighbor-node sets				Enhanced CRO with both sets				Gap (%)
		Best	Average	CPU	SD	Best	Average	CPU	SD	
30	10	33.28620	34.68218	8.48	0.58	33.28620	33.63560	8.63	0.57	−3.02
	20	22.28740	24.68266	8.97	1.02	22.28620	23.13470	9.13	0.36	−6.27
60	10	99.28680	102.13227	7.19	1.15	98.28200	98.28209	7.36	0.00	−3.77
	20	87.28200	88.23140	7.63	0.59	84.28320	84.43416	7.86	0.35	−4.30
90	10	212.27960	213.07834	8.05	0.60	205.28440	205.48527	8.32	0.40	−3.56
	20	193.28680	195.88248	8.36	1.24	188.28080	189.03335	8.72	0.54	−3.50
120	10	269.27780	271.03248	7.48	0.94	265.28500	266.63470	7.62	0.85	−1.62
	20	247.28560	250.53173	7.85	1.51	243.28020	243.28275	8.13	0.00	−2.89
240	10	695.28440	698.17972	7.30	1.13	687.28560	689.43479	7.45	1.56	−1.25
	20	673.28260	677.93227	7.58	1.74	660.28740	662.48431	7.85	1.25	−2.28
300	10	794.28080	798.13128	6.73	2.59	785.28440	788.93251	6.71	2.49	−1.15
	20	780.28620	786.53125	7.24	1.99	771.28680	773.28434	7.43	0.95	−1.68
Avg				7.74	1.26			7.93	0.78	−2.94

Table 7
Result comparison of constant and varying values of δ.

Settings of δ	Best	Average	CPU	SD
1: Varying	770.28560	772.18386	9.30	1.02
2: Constant (0.5)	771.28500	772.43458	9.35	0.67

−2.94%, whereas the average Gaps obtained with the enhanced CRO with $NodeNeighborC_i$ and $NodeNeighborD_i$ are only −2.42% and −2.23%, respectively.

From Tables 1–6, it can be observed that the CPU time does not increase with network size but rather decreases with network size in general. This is due to the network data, which has a longer average operational time between nodes for a larger network in general. Therefore, for a fixed repositioning duration, as the network size increases, the route length (in terms of the number of nodes visited) is shorter. The number of solutions generated by 2-Opt, and hence the CPU time, is lower.

Comparison between constant and varying values of δ

δ represents the probability of running the intensive search in each iteration (see Section 'Intensive search'). To illustrate the benefit of the use of varying values of δ over iterations, two settings of δ were considered in this section. For the first setting, the value of δ obtained from Eq. (39) is used, which varies linearly from 0 to 1. The second part of the second stage, which considers the intensive search, begins when the number of iterations reaches an *IntensivePoint* of 60,000 (see **Algorithm 2**). *maxIteration* is set at 100,000. For the second setting, a constant value of $\delta = 0.5$ (i.e., an average value of δ used in the first setting) is used for the last 40,000 iterations. The starting point of the intensive search is also set at *IntensivePoint* = 60,000.

The instance used in this section is the sharing network containing 300 stations. A single vehicle with a capacity of 20 bicycles is deployed to carry out the repositioning operation within 8 h (T = 28,800 s). The results are shown in Table 7.

Compared with the results obtained with the constant value of δ, the setting with varying δ values achieves solutions with better quality (from the perspectives of both the best and average objective values obtained from 20 runs), although the SD for the solutions is slightly larger. Therefore, Eq. (39) is used to obtain the varying values of δ in the enhanced CRO, rather than adopting a constant value, to control for the occurrence of intensive search.

Comparison between CPLEX, the original CRO, and the enhanced CRO

For comparison purposes, the scenarios (described in Section 'Effectiveness of incorporating neighbor-node sets into enhanced CRO') solved with the enhanced CRO were also solved with the original CRO and with IBM ILOG CPLEX 12.5 with the default settings. The best and average of the best solutions from 20 runs were used to evaluate the performance of the original CRO and the enhanced CRO. These solutions were achieved by running the heuristics to obtain the same number of solutions as the enhanced CRO without the neighbor-node sets (discussed in Section 'Effectiveness of incorporating neighbor-node sets into enhanced CRO'). A running time of 2 h was imposed on CPLEX. Note that CPLEX only checks the time at certain points, not throughout the program running. Therefore, even after the time limit has been reached, CPLEX may not stop until it gets to the certain point to check the time. For this reason, the running time is more than 2 h for some of the scenarios. With this time limit, a program error caused by insufficient memory was found in some cases. The corresponding results were therefore excluded in this section. The optimality tolerance of the solver was set to 0.01%. The result given by CPLEX is "optimal" if the optimal gap, Gap_{opt}, which is a deviation from optimal of 0.01%; otherwise, it indicates an upper bound on the objective value of an optimal solution.

Tables 8–10 present the results from CPLEX, the original CRO, and the enhanced CRO under different repositioning durations where the optimal objective values are printed in bold. In these tables, Gap indicates the performance of the enhanced CRO relative to those of other discussed methods (in percent). It is the deviation of the result (i.e., the best/average of the best objective values) obtained with the enhanced CRO from the result obtained with the discussed method, divided by the result of the discussed method. Gap_1 and Gap_2 are obtained by comparing the best and average of the best objective values obtained with the enhanced CRO with the best objective value of CPLEX (i.e., the optimal objective value [Opt] or an upper bound [UB]), respectively. Gap_3 shows the performance of the enhanced CRO relative to that of the original CRO based on the average of the best objective values. The CPU (in seconds) denotes the average computing time of 20 runs for the enhanced or original CRO, or the running time of CPLEX.

According to Tables 8–10, in which the results between the original CRO and the enhanced CRO are compared, all negative Gap_3 values indicate that the solution quality of the enhanced CRO has been improved. However, under a fixed repositioning duration, Gap_3 becomes less negative as the size of the instance increases. This means that it is more effective to incorporate neighbor-node sets into the CRO to solve smaller BRPs.

The results from CPLEX and the enhanced CRO are discussed with different repositioning durations. For the short-term repositioning duration (T = 14,400 s) and for most of the small-size and medium-size instances (up to 90 stations), the optimal results were obtained with CPLEX, and the enhanced CRO could always find the optimal solutions within 20 runs. For the results of the scenarios with large-size instances with the same repositioning duration, CPLEX could only provide an upper

Table 8
Result comparison of CPLEX, the original CRO, and the enhanced CRO (T = 14,400 s).

|V|	Q	CPLEX			Original CRO				Enhanced CRO				Gap_1 (%)	Gap_2 (%)	Gap_3 (%)
		Opt/UB	Gap_{opt}	CPU	Best	Average	CPU	SD	Best	Average	CPU	SD			
30	10	68.13860	0.01	479.97	79.13560	82.48416	1.45	1.56	**68.13860**	68.13893	1.66	0.00	0.00	0.00	−17.39
	20	**60.14100**	0.01	339.10	73.14040	77.68425	1.46	2.11	**60.14100**	60.64088	1.69	0.86	0.00	0.83	−21.94
60	10	139.13400	1.07	7210.31	151.12960	157.42801	1.36	2.21	139.13380	139.13380	1.42	0.00	0.00	0.00	−11.62
	20	**126.14400**	0.01	973.42	147.13440	151.93176	1.41	2.29	**126.14400**	126.49316	1.58	0.48	0.00	0.28	−16.74
90	10	**252.14400**	0.01	1006.72	272.11940	276.78323	1.42	2.18	**252.14400**	252.64373	1.58	0.59	0.00	0.20	−8.72
	20	**241.14300**	0.01	3227.79	266.13740	271.13527	1.40	2.14	**241.14280**	241.99316	1.61	0.85	0.00	0.35	−10.75
120	10	314.13600	1.19	7200.75	333.13500	334.37525	1.90	0.76	314.13500	314.13530	1.43	0.00	0.00	0.00	−6.05
	20	**294.14400**	0.01	4581.34	322.13200	328.43104	2.08	2.97	**294.14400**	294.49361	1.48	0.57	0.00	0.12	−10.33
240	10	747.12500	1.63	7204.09	760.13920	764.12882	2.27	1.99	736.14400	737.09073	1.66	0.22	−1.47	−1.34	−3.54
	20	726.14000	1.30	7532.29	754.12780	758.88392	1.56	2.38	723.14280	723.39271	1.49	0.43	−0.41	−0.38	−4.68
300	10	847.13400	1.56	7211.45	858.13260	863.73095	2.64	2.08	837.14340	841.33272	1.17	2.49	−1.18	−0.68	−2.64
	20	839.13400	1.75	7394.95	853.13320	860.88008	2.68	2.47	828.14340	828.59298	1.25	0.50	−1.31	−1.26	−3.82
Avg				4530.18			1.80	2.09			1.50	0.58	−0.36	−0.16	−9.85

Table 9
Result comparison of CPLEX, the original CRO, and the enhanced CRO (T = 21,600 s).

|V|	Q	CPLEX			Original CRO				Enhanced CRO				Gap_1 (%)	Gap_2 (%)	Gap_3 (%)
		Opt/UB	Gap_{opt}	CPU	Best	Average	CPU	SD	Best	Average	CPU	SD			
30	10	**47.21420**	0.01	108.16	69.16080	71.44923	3.70	1.79	**47.21420**	47.31414	4.44	0.30	0.00	0.21	−33.78
	20	**39.21420**	0.01	4404.44	59.20640	63.99773	3.74	2.20	**39.21420**	39.26405	4.66	0.22	0.00	0.13	−38.65
60	10	**115.21400**	0.00	15.93	142.19080	148.34749	3.24	2.32	115.21420	115.21432	3.81	0.00	0.00	0.00	−22.33
	20	**103.21600**	0.00	6935.69	132.21420	140.05025	3.41	3.67	104.21480	104.66486	4.02	0.59	0.97	1.40	−25.27
90	10	231.20800	1.74	7205.56	259.20460	266.54824	3.52	3.21	231.20760	231.20778	4.21	0.00	0.00	0.00	−13.26
	20	216.21500	3.17	7203.56	255.19380	258.95490	3.61	2.55	215.21180	215.51297	4.36	0.46	−0.46	−0.32	−16.78
120	10	287.21200	0.28	7202.22	321.21300	324.54740	3.48	1.74	287.21240	287.21267	3.94	0.00	0.00	0.00	−11.50
	20	270.21400	2.43	7266.80	302.21300	316.00226	3.42	3.83	267.21540	268.36234	4.08	0.47	−1.11	−0.69	−15.08
240	10	736.15000	3.72	7599.76	748.20820	755.14770	3.34	2.46	715.20700	715.20859	3.77	0.00	−2.84	−2.84	−5.29
	20	–	–	–	740.20760	747.04698	3.41	3.77	690.21360	692.26303	3.93	0.86	–	–	−7.33
300	10	821.21500	1.84	7205.65	849.20400	856.15010	2.76	2.20	813.20220	813.20502	3.31	0.00	−0.98	−0.98	−5.02
	20	870.10000	8.93	7202.59	844.20760	850.70280	2.94	2.29	800.21240	800.81228	3.60	0.49	−8.03	−7.96	−5.86
Avg				5668.21			3.38	2.67			4.01	0.28	−1.13	−1.00	−16.68

Table 10
Result comparison of CPLEX, the original CRO, and the enhanced CRO (T = 28,800 s).

|V|	Q	CPLEX			Original CRO				Enhanced CRO				Gap_1 (%)	Gap_2 (%)	Gap_3 (%)
		Opt/UB	Gap_{opt}	CPU	Best	Average	CPU	SD	Best	Average	CPU	SD			
30	10	**33.28560**	0.01	1354.15	53.25800	59.20344	6.94	2.29	33.28620	33.63560	8.63	0.57	0.00	1.05	−43.19
	20	**22.28620**	0.01	308.66	46.27060	50.46730	7.30	2.34	**22.28620**	23.13470	9.13	0.36	0.00	3.81	−54.16
60	10	**97.28800**	0.01	2858.53	132.22800	138.36838	6.10	3.00	98.28200	98.28209	7.36	0.00	1.02	1.02	−28.97
	20	84.28620	2.66	7205.01	121.28560	129.51643	6.37	3.61	84.28320	84.43416	7.86	0.35	0.00	0.18	−34.81
90	10	205.28400	0.78	7205.92	249.25320	256.86289	6.74	3.29	205.28440	205.48527	8.32	0.40	0.00	0.10	−20.00
	20	191.28100	4.38	7204.19	241.27240	246.51490	6.92	2.73	188.28080	189.03335	8.72	0.54	−1.57	−1.18	−23.32
120	10	263.28500	0.43	8462.72	311.28800	316.51460	6.29	2.47	265.28500	266.63470	7.62	0.85	0.76	1.27	−15.76
	20	243.28700	2.45	7207.96	300.22680	307.32066	6.46	3.22	243.28020	243.28275	8.13	0.00	0.00	0.00	-20.84
240	10	688.28700	0.78	7203.03	741.28200	746.62153	6.24	2.15	687.28560	689.43479	7.45	1.56	−0.15	0.17	−7.66
	20	–	–	–	726.24540	736.72177	6.18	3.50	660.28740	662.48431	7.85	1.25	–	–	−10.08
300	10	801.26100	2.67	7206.02	842.28080	847.57012	5.47	2.49	785.28440	788.93251	6.71	2.49	−1.99	−1.54	−6.92
	20	803.27200	5.15	9872.86	831.25920	839.91355	5.80	3.58	771.28680	773.28434	7.43	0.95	−3.98	−3.73	−7.93
Avg				6008.10			6.40	2.89			7.93	0.78	−0.54	0.10	−22.80

Table 11
Sensitivity test on μ.

μ	CPLEX			Enhanced CRO				Gap_1 (%)	Gap_2 (%)
	Opt/UB	Gap_{opt}	CPU	Best	Average	CPU	SD		
0.0000	68.00000	0.00	208.42	68.00000	68.00000	1.71	0.00	0.00	0.00
0.0003	72.15800	0.00	403.37	72.15800	72.16610	1.90	0.02	0.00	0.00
0.0006	76.30860	0.00	987.26	76.31600	76.33220	1.92	0.03	0.00	0.00
0.0009	80.47400	0.00	299.37	80.47400	80.50640	1.91	0.06	0.00	0.00
0.0012	84.63200	0.00	254.43	84.63200	84.70400	1.89	0.10	0.00	0.00
0.0015	88.79000	0.00	234.67	88.79000	88.95650	1.88	0.12	0.00	0.00
0.0018	92.94800	0.00	237.61	92.94800	93.02360	1.87	0.13	0.00	0.00
0.0021	97.10600	0.00	109.22	97.10600	97.27530	1.86	0.18	0.00	0.00
0.0024	101.10400	0.00	59.45	101.10400	101.27440	1.85	0.19	0.00	0.00
0.0027	104.99200	0.00	45.47	104.99200	105.24900	1.84	0.15	0.00	0.00
0.0030	108.62000	0.00	5.27	108.88000	109.09000	1.84	0.12	0.00	0.00
Avg		0.00	258.60			1.86	0.10	0.00	0.00

bound for the problem in most cases. The negative Gap_1 values for most of these scenarios indicate that a better upper bound is found by the enhanced CRO. Considering all scenarios, the best and average of the best objective values obtained with the enhanced CRO are −0.36% and −0.16%, respectively, from the corresponding optimal solution or the upper bound on average, which means that the enhanced CRO gives better solutions on average. Moreover, the mean computing time is 1.50 s. Thus, compared with the average computing time of CPLEX (4530.18 s), the enhanced CRO outperforms CPLEX in computation time.

From Table 9 (T = 21,600 s), the average objective values obtained with the enhanced CRO show that it could achieve optimal solutions for three scenarios, provide a good upper bound for three scenarios, and improve the upper bounds obtained with CPLEX for five scenarios within a very short computing time. The average gaps achieved by the best and average of the best objective values obtained with the enhanced CRO are −1.13% and −1.00%, respectively, which means that the solution quality of the enhanced CRO is still better when T = 21,600 s.

For the scenarios where the repositioning duration T = 28,800 s, the enhanced CRO could achieve optimal solutions for two scenarios, good upper bounds for three scenarios, one upper bound better than those of CPLEX for the small-size to medium-size instances, and two good upper bounds and two upper bounds better than those of CPLEX for large-size instances (with at least 120 stations). The average computing time was 7.93 s, which is also very short compared with that of CPLEX (6008.10 s).

Regardless of the repositioning duration, CPLEX could not always find optimal solutions for the large-size instances. In our study, when $|V|$ reached 240, CPLEX could not find feasible solutions within the 2-h time limit for the case of T = 21,600 s and Q = 20 and the case of T = 28,800 s and Q = 20. Unlike CPLEX, the enhanced CRO always found feasible solutions quickly, even when the sizes of the test instances were large. Moreover, the enhanced CRO always outperformed CPLEX in terms of solution quality when solving large instances and obtained good upper bounds or achieve optimality when solving other instances. Therefore, we can conclude that the enhanced CRO always obtains good feasible solutions much more quickly than CPLEX.

Sensitivity test on μ

μ determines the relative importance between the total unmet demand reflected by $\sum_{i\in V}\psi_i$ and the vehicle's total operational time $\sum_{i,j\in V_0, i\neq j} t_{ij}\cdot x_{ij}$. In this section, the sensitivity of μ is tested with a sharing network containing 30 stations. Repositioning was carried out by a single vehicle with a capacity of 10 bicycles within 4 h. Different values of μ from 0 to 0.003 in increments of 0.0003 were considered. The results obtained with IBM ILOG CPLEX 12.5 and the enhanced CRO were compared. CPLEX was performed with its default settings and a 2-h running time limit, and the enhanced CRO was executed with a *maxIteration* of 100,000. To evaluate the performance of the enhanced CRO, the best and average of the best solutions obtained from 20 runs were used. The results are shown in the table below. CPU is the running time for CPLEX (in seconds) or the average computing time of 20 runs for the enhanced CRO.

According to Table 11, the fact that all values of optimal gap, Gap_{opt}, are zero indicates that all of the results obtained with CPLEX are optimal solutions. Gap_1 and Gap_2 are the deviations of the best/average of the best objective values from 20 runs obtained with the enhanced CRO from the corresponding optimal solution obtained with CPLEX, respectively. The best objective values obtained with the enhanced CRO show that the enhanced CRO can always obtain optimal solutions under different values of μ within 20 runs except for the cases $\mu = 0.0006$ and $\mu = 0.0030$. The average of the best objective values obtained with the enhanced CRO from 20 runs for each of the discussed cases indicates that nearly optimal solutions are obtained with the enhanced CRO in general. Moreover, the average computing time of the enhanced CRO is 1.86 s (SD = 0.10), which is much shorter than that of CPLEX. Therefore, we can conclude that the enhanced CRO works well under the different values of μ.

Conclusions

An enhanced version of CRO for a static repositioning problem with a single vehicle is proposed. The enhanced CRO incorporates novel operators, new rules, and an intensive search to improve the solution quality. It also considers station characteristics and proposes two neighbor-node sets to narrow the search space. It is mainly used to deal with vehicle routes. A subroutine is proposed and incorporated within the enhanced CRO to determine the loading and unloading quantities at each visited station. This determination method is different from and more efficient than other methods (e.g., Rainer-Harbach et al., 2015) that rely on solving the subproblem as a linear programming or max-flow subproblem in each iteration. The computational experiments were conducted on instances up to 300 stations. The computational results indicate that the enhanced CRO provides high-quality solutions with short computing times. Specially, it outperforms the original CRO in terms of solution quality and CPLEX in terms of speed; it can always outperform CPLEX in terms of solution quality when solving large instances. The results also confirm that incorporation of the two neighbor-node sets into CRO is beneficial and that the probability of running the intensive search should increase with the iteration number in the final part of the main stage of the enhanced CRO.

Acknowledgements

This research was supported by a grant from the National Natural Science Foundation of China (71271183). The authors are grateful to the three reviewers for their constructive comments.

References

Bai, L., Liu, P., Chen, Y., Zhang, X., Wang, W., 2013. Comparative analysis of the safety effects of electric bikes at signalized intersections. Transp. Res. Part D: Transp. Environ. 20, 48–54.
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 – Oper. 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. Transp. Res. Procedia, vol. 10, pp. 364–373.
Buehler, R., 2012. Determinants of bicycle commuting in the Washington, DC region: The role of bicycle parking, cyclist showers, and free car parking at work. Transp. Res. Part D: Transp. Environ. 17 (7), 525–531.
Caggiani, L., Ottomanelli, M., 2012. A modular soft computing based method for vehicles repositioning in bike-sharing systems. Procedia – Soc. Behav. Sci. 54, 675–684.
Castillo-Manzano, J.I., Castro-Nuño, M., López-Valpuesta, L., 2015. Analyzing the transition from a public bicycle system to bicycle ownership: A complex relationship. Transp. Res. Part D: Transp. Environ. 38, 15–26.
Caulfield, B., Brick, E., McCarthy, O.T., 2012. Determining bicycle infrastructure preferences—A case study of Dublin. Transp. Res. Part D: Transp. Environ. 17 (5), 413–417.
Chemla, D., Meunier, F., Wolfler Calvo, R., 2013. Bike sharing systems: Solving the static rebalancing problem. Discrete 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.
Dell'Amico, M., Hadjicostantinou, E., Iori, M., Novellani, S., 2014. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. Omega 45, 7–19.
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, vol. 8124. 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, vol. 7919. Springer, Berlin Heidelberg, pp. 198–212.
Erdoğan, G., Battarra, M., Wolfler Calvo, R., 2015. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. Eur. J. Oper. Res. 245 (3), 667–679.
Erdoğan, G., Laporte, G., Wolfler Calvo, R., 2014. The static bicycle relocation problem with demand intervals. Eur. J. Oper. Res. 238 (2), 451–457.
Fishman, E., Washington, S., Haworth, N., 2014a. Bike share's impact on car use: Evidence from the United States, Great Britain, and Australia. Transp. Res. Part D: Transp. Environ. 31, 13–20.
Fishman, E., Washington, S., Haworth, N., Mazzei, A., 2014b. Barriers to bikesharing: An analysis from Melbourne and Brisbane. J. Transp. Geogr. 41, 325–337.
Forma, I., Raviv, T., Tzur, M., 2015. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. Transp. Res. Part B: Methodol. 71, 230–247.
Fricker, C., Gast, N., 2014. Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. EURO J. Transp. Logist., 1–31
Fürst, E., 2014. Making the way to the university environmentally sustainable: A segmentation approach. Transp. Res. Part D: Transp. Environ. 31, 1–12.
Fyhri, A., Fearnley, N., 2015. Effects of e-bikes on bicycle use and mode share. Transp. Res. Part D: Transp. Environ. 36, 45–52.
Goodman, A., Cheshire, J., 2014. Inequalities in the London bicycle sharing system revisited: Impacts of extending the scheme to poorer areas but then doubling prices. J. Transp. Geogr. 41, 272–279.
Hernández-Pérez, H., Salazar-González, J.-J., 2004. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discr. Appl. Math. 145 (1), 126–139.
Hernández-Pérez, H., Salazar-González, J.-J., 2007. The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. Networks 50 (4), 258–272.
Ho, S.C., Szeto, W.Y., 2014. Solving a static repositioning problem in bike-sharing systems using iterated tabu search. Transp. Res. Part E: Logis. Transp. Rev. 69, 180–198.
Ho, S.C., Szeto, W.Y., 2016. GRASP with path relinking for the selective pickup and delivery problem. Expert Syst. Appl. 51 (1), 14–25.
Kitthamkesorn, S., Chen, A., Xu, X., Ryu, S., 2016. Modeling mode and route similarities in network equilibrium problem with go-green modes. Networks Spatial Econ. 16 (1), 33–60.
Kloimüllner, C., Papazek, P., Hu, B., Raidl, G.R., 2014. Balancing bicycle sharing systems: An approach for the dynamic case. In: Blum, C., Ochoa, G. (Eds.), Evolutionary Computation in Combinatorial Optimisation, vol. 8600. Springer, Berlin Heidelberg, pp. 73–84.
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. Evol. Comput. 14 (3), 381–399.
Lam, A.Y.S., Li, V.O.K., Wei, Z., 2012. Chemical reaction optimization for the fuzzy rule learning problem. In: Proceedings of IEEE Congress on Evolutionary Computation (IEEE CEC 2012), Brisbane, Australia.
Lin, S., 1965. Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. 44 (10), 2245–2269.
Lu, C.-C., 2016. Robust multi-period fleet allocation models for bike-sharing systems. Networks Spatial Econ. 16 (1), 61–82.
Meddin, R., DeMaio, P., 2015. The bike-sharing world map. <http://www.bikesharingworld.com/> (access on 12 April).
Nair, R., Miller-Hooks, E., 2011. Fleet management for vehicle sharing operations. Transp. Sci. 45 (4), 524–540.
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.), Parallel Problem Solving from Nature – PPSN XIII, vol. 8672. Springer International Publishing, pp. 792–801.
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, vol. 8111. Springer, Berlin Heidelberg, pp. 372–379.
Pfrommer, J., Warrington, J., Schildbach, G., Morari, M., 2014. Dynamic vehicle redistribution and online price incentives in shared mobility systems. IEEE Trans. Intell. Transp. Syst. 15 (4), 1567–1578.
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. Global 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. Transp. Logist. 2 (3), 187–229.
Regue, R., Recker, W., 2014. Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem. Transp. Res. Part E: Logist. Transp. Rev. 72, 192–209.
Ricci, M., 2015. Bike sharing: A review of evidence on impacts and processes of implementation and operation. Res. Transp. Bus. Manage. 15, 28–38.
Ruch, C., Warrington, J., Morari, M., 2014. Rule-based price control for bike sharing systems. In: Control Conference (ECC), 2014 European. IEEE, pp. 708–713.
Schuijbroek, J., Hampshire, R., van Hoeve, W.-j., 2013. Inventory rebalancing and vehicle routing in bike sharing systems (Report No. 1491). Tepper School of Business. <http://repository.cmu.edu/tepper/1491/>.
Singla, A., Santoni, M., Bartók, G., Mukerji, P., Meenen, M., Krause, A., 2015. Incentivizing users for balancing bike sharing systems. In: Paper presented at Twenty-Ninth AAAI Conference on Artificial Intelligence.
Shaheen, S. A., Martin, E. W., Chan, N. D., Cohen, A. P., Pogodzinski, M., 2014. Public bikesharing in North America during a period of rapid expansion: Understanding business models, industry trends and user impacts (CA-MTI-14-1131). <http://transweb.sjsu.edu/PDFs/research/1131-public-bikesharing-business-models-trends-impacts.pdf> (access on 31 March 2016).
Szeto, W.Y., Wang, Y., Wong, S.C., 2014. The chemical reaction optimization approach to solving the environmentally sustainable network design problem. Comput.-Aided Civ. Infrastruct. Eng. 29 (2), 140–158.
Thigpen, C.G., Driller, B.K., Handy, S.L., 2015. Using a stages of change approach to explore opportunities for increasing bicycle commuting. Transp. Res. Part D: Transp. Environ. 39, 44–55.
Ting, C.-K., Liao, X.-L., 2013. The selective pickup and delivery problem: Formulation and a memetic algorithm. Int. J. Prod. Econ. 141 (1), 199–211.
Xu, J., Lam, A.Y.S., Li, V.O.K., 2011. Stock portfolio selection using chemical reaction optimization. In: Proceedings of International Conference on Operations Research and Financial Engineering (ICORFE 2011), Paris, France, pp. 458–463.
Yu, J.J.Q., Li, V.O.K., Lam, A.Y.S., 2012. Sensor deployment for air pollution monitoring using public transportation system. In: Proceedings of IEEE Congress on Evolutionary Computation (IEEE CEC 2012), Brisbane, Australia.