Transportation Research Part C 80 (2017) 92–116

Contents lists available at ScienceDirect

Transportation Research Part C

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

Free-floating bike sharing: Solving real-life large-scale static rebalancing problems ☆

Aritra Pal [a], Yu Zhang [b,c,*]

[a] Department of Industrial and Management Systems Engineering, University of South Florida, United States
[b] Department of Civil and Environmental Engineering, University of South Florida, United States
[c] College of Transportation Engineering, Tongji University, China

ARTICLE INFO

Article history:
Received 24 April 2016
Received in revised form 15 March 2017
Accepted 19 March 2017
Available online 3 May 2017

Keywords:
Free-floating bike sharing
Pickup and delivery
Granular neighborhoods
Variable neighborhood descent
Large neighborhood search

ABSTRACT

Free-floating bike sharing (FFBS) is an innovative bike sharing model. FFBS saves on start-up cost, in comparison to station-based bike sharing (SBBS), by avoiding construction of expensive docking stations and kiosk machines. FFBS prevents bike theft and offers significant opportunities for smart management by tracking bikes in real-time with built-in GPS. However, like SBBS, the success of FFBS depends on the efficiency of its rebalancing operations to serve the maximal demand as possible.

Bicycle rebalancing refers to the reestablishment of the number of bikes at sites to desired quantities by using a fleet of vehicles transporting the bicycles. Static rebalancing for SBBS is a challenging combinatorial optimization problem. FFBS takes it a step further, with an increase in the scale of the problem. This article is the first effort in a series of studies of FFBS planning and management, tackling static rebalancing with single and multiple vehicles. We present a Novel Mixed Integer Linear Program for solving the Static Complete Rebalancing Problem. The proposed formulation, can not only handle single as well as multiple vehicles, but also allows for multiple visits to a node by the same vehicle. We present a hybrid nested large neighborhood search with variable neighborhood descent algorithm, which is both effective and efficient in solving static complete rebalancing problems for large-scale bike sharing programs.

Computational experiments were carried out on the 1 Commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP) instances used previously in the literature and on three new sets of instances, two (one real-life and one general) based on Share-A-Bull Bikes (SABB) FFBS program recently launched at the Tampa campus of University of South Florida and the other based on Divvy SBBS in Chicago. Computational experiments on the 1-PDTSP instances demonstrate that the proposed algorithm outperforms a tabu search algorithm and is highly competitive with exact algorithms previously reported in the literature for solving static rebalancing problems in SBSS. Computational experiments on the SABB and Divvy instances, demonstrate that the proposed algorithm is able to deal with the increase in scale of the static rebalancing problem pertaining to both FFBS and SBBS, while deriving high-quality solutions in a reasonable amount of CPU time.

© 2017 Elsevier Ltd. All rights reserved.

☆ This article belongs to the Virtual Special Issue on "Emerging Urban Mobility Services: Characterization, Modeling and Application".

* Corresponding author at: Department of Civil and Environmental Engineering, University of South Florida, United States.
E-mail address: yuzhang@usf.edu (Y. Zhang).

http://dx.doi.org/10.1016/j.trc.2017.03.016
0968-090X/© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

Bike sharing allows people a healthy, enjoyable and emission-free way to commute across small distances free from the worries of owning a bike. It also provides an alternative and attractive solution for the first and last-mile problem in multi-modal transportation. Over the years, various schemes of bike sharing have been presented, with the earliest generation dating back to July, 1965, in Amsterdam with Witte Fietsen (White Bikes). The next generation, coin-deposit systems, first were introduced in Denmark in Farse and Grena in 1991 and then in Nakskov in 1993. A major breakthrough came when people could use a magnetic stripe card to rent a bike. This generation of bike sharing, known as the IT-based system started with Bikeabout in 1996 at Portsmouth University, England. Interested readers are referred to DeMaio (2009), for an overview of various generations of bike sharing.

Free-floating bike sharing (FFBS) is relatively new. In this model of bike sharing, bikes can be locked to an ordinary bicycle rack (or any solid frame or standalone), thus eliminating the need for specific stations. In comparison to the prevailing Station-based Bike Sharing (SBBS), FFBS saves on start-up cost by avoiding the construction of expensive docking stations and kiosk machines. By tracking bikes in real-time with built-in GPS, FFBS prevents bike theft, and offers significant opportunities for smart management. With FFBS, customer satisfaction levels increase because obtaining and returning the bikes becomes much more convenient compared to SBBS; the average walking distance of FFBS is shorter and customers do not have to worry about the shortage of vacant spots at stations where they need to return the bikes.

SocialBicycles (SoBi) is one of the providers of FFBS bikes. SoBi bikes are used as an example to further illustrate how FFBS works. Each registered SoBi member gets a unique PIN and can use a smartphone app to locate available bikes. After reserving a bike, the user has 15 min to get to its location. Once the user finds the bike, (s)he enters the PIN on the bike's built-in keypad to unlock the bike. If the user wants to stop somewhere quickly, the bike can be locked and placed on hold. Upon reaching the destination, the user can simply lock the bike to a bicycle rack (or any solid frame or standalone) and the bike becomes available for the next user (see Fig. 1).

During daily operation, the distribution of bikes in the system becomes skewed, often leading to a low quality of service and user dissatisfaction. To prevent such a scenario from prevailing, operators move bikes across the network to achieve a desired distribution. The operation of redistributing bikes across the network using a fleet of vehicle(s) is known as bike rebalancing. Rebalancing at night, when user intervention is negligible, is called static rebalancing. If user intervention is considered, the problem is called dynamic rebalancing. Bike rebalancing being a variant of vehicle routing problem, is a challenging combinatorial optimization problem, with the objective function being, to minimize the financial and environmental costs of rebalancing. Different variants of the bike rebalancing problem have been proposed in the literature, see Section 2 for a detailed literature review.

Dependent on how rigorous of a rebalancing needs to be performed, it can be classified into two categories, complete and partial rebalancing. In complete rebalancing the rebalancing operation terminates only when target inventory of all nodes in the network have been met. However, if complete rebalancing is not feasible (for example: if the time taken to completely rebalance the bike sharing system is more than the actual time available for rebalancing), the operator may consider partial rebalancing. In partial rebalancing, not all nodes will meet their target inventory. In SBBS, nodes are working stations with status of having a deficit or surplus of bikes, or being self-balanced. FFBS has no stations like in SBBS, so nodes in FFBS include regular bike racks and standalone locations where bikes are parked by users, and bike racks and standalone locations where

Fig. 1. A Share-A-Bull Bike locked to a wooden frame.

bikes have not been parked but are perceived as important locations for bikes to be present by the operator. For FFBS, there are no stations like in SBBS. In this study, we focus on Static Complete Rebalancing Problem (SCRP). Partial rebalancing are an on-going effort of our research team and will be addressed in a future article. The static complete rebalancing problem (SCRP) is computationally more challenging than static partial rebalancing problem, because the number of times a node is visited, in the optimal solution cannot be determined apriori. We are also performing studies on understanding demand patterns of bike sharing system and explore how dynamic rebalancing can be applied in real world cases.

For the same configuration, i.e., number of stations (in case of SBBS and bike racks in case of FFBS), number of bikes and capacity of the rebalancing fleet, computational complexity of SCRP is higher for FFBS than for SBBS. To illustrate this, let us consider a bike sharing system with 100 stations (or bike racks) and 200 bikes. In case of SBBS, number of locations that the rebalancing vehicle(s) has to visit to completely rebalance the system will at most be 100. This is because some stations may be self rebalanced. However, in case of FFBS, number of locations that the rebalancing vehicle(s) has to visit to completely rebalance the system can at times be $\gg 100$. To illustrate this, let us consider the scenario, when all 200 bikes are parked outside of bike racks in standalone locations but the operator wants each bike rack to have 2 bikes each. In this scenario, number of locations that the rebalancing vehicle(s) has to visit to completely rebalance the system is 300, out of which 200 are standalone locations (for pickup) where bikes are parked and 100 are bike racks (for drop offs). Now, let us consider the scenario when instead of 100 bike racks, the system has 300 bike racks and all 200 bikes are parked outside of bike racks in standalone locations. In this case the operator can have at most 200 bike racks to be filled with 1 bike each. In this scenario, number of locations that the rebalancing vehicle(s) has to visit to completely rebalance the system is at most 400, out of which 200 are standalone locations (for pickup) where bikes are parked and 200 out of 300 bike racks where at least 1 bike needs to be dropped off. Thus we can conclude that nodes in the system that a rebalancing vehicle(s) has to visit is $\leqslant$*Number of Working Stations* in case of SBBS and is $\leqslant$*min {Number of Bike Racks + Number of Bikes,* $2\times$ *Number of Bikes*} in case of FFBS. This fact is also evident from the real life instances introduced in Section 7.

Chemla et al. (2013a) was the first to introduce SCRP for SBBS and proposed tabu search algorithms for solving it. Erdoan et al. (2015) proposed a time extended network formulation of SCRP for SBBS and solved it exactly using mixed integer programming. However, the mathematical formulations proposed in Chemla et al. (2013a) and Erdoan et al. (2015) can handle only a single vehicle. Further, the time extended network formulation of SCRP were designed in a manner that it could not be extended for multiple vehicles. This is evident when in Alvarez-Valdes et al. (2016) a heuristic method is proposed for solving SCRP in SBBS for a fleet of multiple vehicles, however the authors are unable to present any mathematical formulation. This issue is being addressed by a mathematical formulation based on spacial decomposition of the network into nodes of unit imbalance each, except for the depot whose imbalance is 0. The proposed formulation, can not only handle single and multiple vehicles, but also allows for multiple visits to a node by the same vehicle. For more details on the proposed formulation see Section 3.

Tabu search and exact algorithms proposed in Chemla et al. (2013a) and Erdoan et al. (2015) respectively, are not effective for solving static rebalancing problems even in small or medium scale FFBS or SCRP with multiple vehicles. Thus, in this study a heuristic algorithm is proposed, to derive high quality solutions of SCRP with both single as well as multiple vehicles, in a reasonable amount of CPU time. The proposed heuristic consists of creating an initial solution using a greedy construction heuristic and improving it until no further improvement is possible. The improvement heuristic is a hybridization of variable neighborhood descent with large neighborhood search. Seven granular descent operators (Toth and Vigo, 2003) are used for variable neighborhood descent. Four perturbation and three repairing operators were developed, resulting in a total of twelve large neighborhoods. Each of these is explored exhaustively before moving on to the next large neighborhood until no further improvement is possible, resulting in a nested large neighborhood search. For more details on the proposed heuristic see Section 4.

Computational experiments on the 1-PDTSP instances from the literature, demonstrate that the presented algorithm outperforms the tabu search algorithm (Chemla et al., 2013a) and is highly competitive with the exact algorithms (Erdoan et al., 2015) for solving SCRP in SBBS. It is able to find new solutions for 59 of 148 instances for which the optimal solution is not known and on average 400 and 36 times faster than the exact and the tabu search algorithms proposed in the literature. Computational experiments on the new SABB FFBS instances (consisting of up to 400 nodes, 300 bikes, and a eet size of up to 3 vehicles) and Divvy instances (consisting of 450 stations, 3000 bikes, and a size of up to 30 vehicles), demonstrate that NLNS + VND is able to deal with the increase in scale of SCRP for both FFBS and SBBS. It also shows that SCRP is feasible for both SABB program at USF, Tampa and Divvy SBBS at Chicago with the given size of the rebalancing fleet.

The remainder of the paper is organized as follows. Section 2 describes SCRP in detail and presents the literature review of rebalancing operations on bike sharing systems. Section 3 describes the mathematical formulation proposed for SCRP. Section 4 describes our proposed heuristic for deriving high quality solutions of SCRP. Section 5 discusses our recommended strategies for solving different types of SCRP using our proposed methodology. Sections 6–8 summarizes the experimental results and conclusions of the three case studies. Section 9 concludes the paper with directions for future research.

2. Problem description and related work

In recent years, with the boom of SBBS, extensive bike-share related research has been conducted and documented. Related to the operational management of a bike-sharing system, the literature can be grouped into three major research

sub-streams: demand analysis, service-level analysis and rebalancing strategies. Service-level and demand Analysis are beyond the scope of this study and will be summarized in a future article we are working on. In this article, we focus only on the literature relevant to rebalancing operations. Rebalancing a bike sharing system can be achieved in various ways, illustrated in Fig. 2. In Fig. 2, operator and user based strategies refers to manually rebalancing the bikes in the network using a fleet of rebalancing trucks and incentivizing users to encourage them to self-rebalance the bikes in the network respectively. If the manual rebalancing is done when the user intervention is negligible, it is known as static rebalancing, whereas if it is done when there is significant user intervention, the rebalancing is known as dynamic rebalancing. Further, in SCRP the operator meets the target inventory level at all the nodes in the network exactly. However, if the resources available (rebalancing time or number of rebalancing vehicles) to the operator is not sufficient for complete rebalancing, partial target inventory levels are met at certain or at all the nodes. This is known as partial rebalancing (SPRP).

A summary of the recent literature of operator based rebalancing strategies in SBBS is provided in Table 1. In terms of user based rebalancing strategies for SBBS, Chemla et al. (2013b) and Pfrommer et al. (2014) presented dynamic pricing

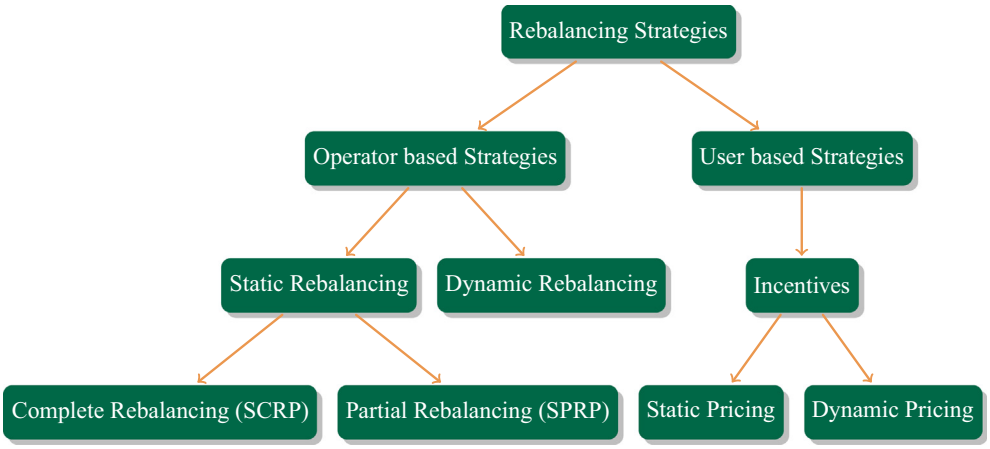

Fig. 2. Subdivisions of rebalancing strategies.

Table 1
Summary of literature of operator based rebalancing strategies.

Research article	Type of rebalancing	Subtype	Fleet size	Stations	Methodology
Chemla et al. (2013a)	Static rebalancing	Complete rebalancing	1	100	Tabu Search
Erdoan et al. (2015)			1	60	Exact algorithm based on Bender's Cuts
Alvarez-Valdes et al. (2016)			2	28	Heuristic based on Minimum Cost Flow
Raviv et al. (2013)		Partial rebalancing	2	60	Mixed Integer Programming (MIP)
Forma et al. (2015)			3	200	3-Step Matheuristic
Ho and Szeto (2014)			1	400	Iterated Tabu Search
Schuijbroek et al. (2017)			5	135	Constraint Programming and MIP
Erdoan et al. (2014)			1	50	Branch and Cut and Bender's Decomposition
Dell'Amico et al. (2014)			1	116	Branch and Cut
DellAmico et al. (2016)			1	500	Metaheuristic based on Destroy and Repair
Rainer-Harbach et al. (2015)			21	700	Combination of Greedy Heuristics, GRASP and VNS
Szeto et al. (2016)			1	300	Chemical Reaction Optimization
Ho and Szeto (2017)			5	518	Hybrid Large Neighborhood Search
Pfrommer et al. (2014)	Dynamic rebalancing		1	–	Greedy Heuristics
Contardo et al. (2012)			100	A hybrid MIP approach using Dantzig-Wolfe and Benders decomposition	
Regue and Recker (2014)			–	MIP	
Kloimüllner et al. (2014)			90	Combination of Greedy Heuristics, GRASP and VNS	

strategies, which encourage users to return bikes to empty (or unsaturated) nodes. Singla et al. (2015) extended the model of Pfrommer et al. (2014) by incorporating a learning mechanism to shape the user utility function, and enriched it by taking into account a budget constraint for the operator. For a more detailed literature review of rebalancing strategies in station based shared mobility systems, the readers are referred to Laporte et al. (2015). From the literature review, it can be concluded that only Reiss and Bogenberger (2015) reported a study conducted on FFBS. Weikl and Bogenberger (2013) and Boyacı et al. (2015) report SPRP schemes for one-way station-based (SBCS) and Free Floating Car Sharing (FFCS) systems respectively. However, no article has reported any SCRP scheme for either FFBS or FFCS.

In this study, the objective function of SCRP is minimizing the makespan of the fleet of rebalancing vehicles. This is equivalent to minimizing the maximum rebalancing time of the fleet of rebalancing vehicles, as is done in Schuijbroek et al. (2017). However, if the fleet consist of a single vehicle, it is equivalent to minimizing the total distance traversed by the rebalancing vehicle, as is done in Chemla et al. (2013a) and Erdoan et al. (2015). Different studies in the literature have minimized different objective functions, including weighted sum of two or more measures. The most important measures being:

1. Travel cost
2. Total redistribution (travel + loading and unloading) cost
3. Total absolute deviation from the target number of bikes
4. Total unmet customer demand or a penalty cost
5. Makespan of the rebalancing fleet – Measure used in this study

For SCRP, options 3 and 4 are not applicable as the system is completely rebalanced. In this study, the objective function of SCRP is to minimize the make-span of the fleet of rebalancing vehicles. If the fleet consists of a single vehicle, it is equivalent to minimizing the total distance (travel cost as well) traversed by the rebalancing vehicle. If multiple vehicles are needed, minimizing the make-span is better than minimizing the total travel distance (travel cost) because the latter may create problems that one vehicle is allotted a rebalancing trip whose time is significantly longer than that of another vehicle. Minimizing make-span determines the needed time allotted to bike rebalancing and make sure that rebalancing workload more evenly distributed to multiple vehicles. It decreases the variance of the rebalancing time of the fleet while at the same time minimizing the overall rebalancing time to a great extent.

Preemption is not allowed in the tours of the rebalancing vehicles, which means that nodes cannot be used as buffers for storing bikes. For more details pertaining to preemption in SCRP, the readers are referred to Chemla et al. (2013a) and Erdoan et al. (2015). Allowing preemption only increases the computational complexity of the mixed integer linear program (Section 3) and the solution algorithm (Section 4) without much improvement in solution quality. The computational complexity increases because the number of (possible) nodes to visit increases when preemption is allowed. If preemption is not allowed, nodes which are already balanced can be removed in the preprocessing phase. The solution space also decreases because the inventory level can only either increase or decrease monotonically from the initial inventory levels to the target inventory levels. Erdoan et al. (2015) empirically showed that, preemption adds a value of 0.6%, at most in the solution quality, for the 1-PDTSP instances used in the literature.

Mathematical formulations proposed in the literature for SCRP can only handle a single vehicle. The formulations were designed in a manner that they could not be extended for multiple vehicles. This is evident when in Alvarez-Valdes et al. (2016) a heuristic method is proposed for SCRP in SBBS using multiple vehicles, however the authors are unable to present any mathematical formulation of SCRP with multiple vehicles. This issue is addressed by introducing a mathematical formulation based on spacial decomposition of the network into nodes of unit imbalance each, except for the depot whose imbalance is 0. Our formulation, can not only handle single and multiple vehicles, but also allows for multiple visits to a node by the same vehicle. In the existing literature, to deal with the scale of the static rebalancing problem, researchers have sacrificed solution quality by limiting the number of visits to a node to, at most, once. If multiple visits to a node are allowed, the solution algorithms are unable to cope with the scale of the problem and become ineffective for nodes greater than 50. The proposed heuristic addresses both of these issues, i.e., retaining solution quality with increase in the scale of the static rebalancing problem while allowing multiple visits to a node. Our solution algorithm can also handle a fleet size of 30 vehicles.

3. Mathematical formulation of SCRP

Bike rebalancing network consists of nodes with non-zero imbalance and a depot with 0 imbalance. Fig. 3 is an example of a network consisting of three nodes, Node 1 or the Depot, Node 2 with a positive imbalance of 2 (at Node 2 there is a surplus of two bikes) and Node 3 with a negative imbalance of 2 (at Node 3 there is a deficit of two bikes). Erdoan et al. (2015) was able to formulate the SCRP for a fleet of a single vehicle using such a network (named original network thereafter). However, it is mathematically challenging to formulate a SCRP with a fleet of multiple vehicles using the original network, as the number of visits to a node by a vehicle is unknown apriori. To address this issue, each node (other than the depot) in the original network is decomposed into nodes each with unit imbalance, but at same geographic location. Fig. 4 is the decomposed network of the original network in Fig. 3. In the decomposed network, Node 2 from the original network is decomposed into two nodes 2_1 and 2_2, with the same geographic location as Node 2 but each with a positive unit imbalance. Similarly, Node 3 from

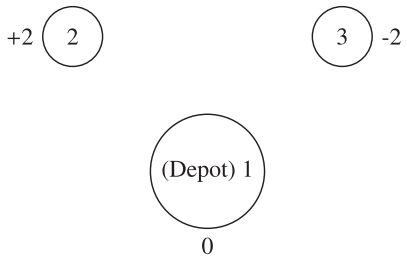

Fig. 3. Original network.

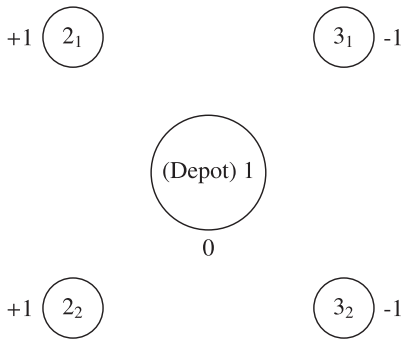

Fig. 4. Decomposed network.

the original network is decomposed into two nodes 3_1 and 3_2, with the same geographic location as Node 3 but each with a negative unit imbalance. Table 2 describes the notations used in the rest of the paper.

In the decomposed network, SCRP becomes feasible to formulate, because the number of visits to a node in the decomposed network must equal one. This is because, multiple visits to a node may be required, if and only if its absolute imbalance is strictly greater than one. However, every node in the decomposed network has a **unit absolute imbalance**. Now, SCRP with multiple vehicles can be formulated as a multiple traveling salesman problem (m-TSP) with additional constraints, on the decomposed network. For a detailed overview on m-TSP, readers are referred to Laporte and Nobert (1980) and Bektas (2006).

Let us now consider two other variants of the Traveling Salesman Problem (TSP) – 1 Commodity Pickup and Delivery TSP (1-PDTSP) and Q-TSP or also known as Capacitated TSP with Pickups and Deliveries (CTSPPD). 1-PDTSP (Hernndez-Prez and Salazar-Gonzlez, 2004) is a generalization of TSP, in which nodes (providing or requiring known amounts of a product) must be visited exactly once by a vehicle (with a given capacity) serving the imbalances, while minimizing the total travel distance. Q-TSP (Chalasani and Motwani, 1999) or CTSPPD (Anily and Bramel, 1999) is a special case of 1-PDTSP, where the delivery and pickup quantities are all equal to one unit.

SCRP with a single rebalancing vehicle on the original network is not equivalent to 1-PDTSP, because in 1-PDTSP number of visits to a node is limited to exactly once, irrespective of the imbalances at the node. This limitation on the number of visits to once, makes 1-PDTSP relatively easy to solve compared to SCRP. However, SCRP on the decomposed network with a single vehicle is equivalent to formulating a 1-PDTSP, a Q-TSP or a CTSPPD problem. However, in order to extend the formulation to handle a fleet of (homogeneous) multiple vehicles, our formulation is based on m-TSP rather than on 1-PDTSP, Q-TSP or CTSPPD, as doing so reduces the quantity of decision variables. Thus, our contribution in terms of the mathematical programming formulation is coming up with a simple, yet effective scheme to decompose the original network, so that existing formulations of different variants of TSP and Vehicle Routing Problem (VRP) can be used to formulate SCRP with both single and multiple vehicles.

After a feasible solution of SCRP on the decomposed network is obtained, it can be converted to a feasible solution for SCRP on the original network, by switching the indices of nodes in the tour(s) of the rebalancing vehicles, with their respective indices in the original network. For example, let us consider SCRP with a single vehicle of capacity 2, on the original network in Fig. 3. However, if instead of choosing to use the original network, its corresponding decomposed network (Fig. 4) is used to compute a feasible solution (Fig. 5), it can be converted to a solution feasible on the original network (Fig. 6) by switching the indices of nodes in the tour with their respective indices in the original network. One more thing that can be done, is to combine consecutive locations with same node indices together into one location and their instruction equal to the sum of that of those corresponding locations. Now, let us consider SCRP with a single vehicle of capacity 1, on the original network in Fig. 3. Fig. 7 represents a feasible solution on the decomposed network and Fig. 8 represents the corre-

Table 2
Notations used in the remainder of the paper.

	Notation	Description	Network		MILP	Heuristic
			Original	Decomposed		
Parameters	$\mathcal{G} = (\mathcal{N}, \mathcal{E})$	Graph of the original network	✓	×		✓
	$\mathcal{N}$	Set of nodes in the original network, including the depot. The depot is denoted by 1, where the tour of the rebalancing vehicles start and end. Rest of the nodes are numbered from $2, 3, \ldots, \mathcal{N}$				
	$\mathcal{E}$	Edges in the original network				
	c_{ij}	Time taken to travel from node i to node j, $\forall (i,j) \in \mathcal{E}$ in seconds				
	d_i	Imbalance at node i, $\forall i \in \mathcal{N}$. $d_i > 0$ when node i is a pickup node or has a surplus of bikes, $d_i < 0$ when node i is a delivery node or has a deficit of bikes and 0 when node i is the depot				
	$\mathcal{G}_o = (\mathcal{N}_o, \mathcal{E}_o)$	Graph of the decomposed network	×	✓		×
	$\mathcal{N}_o$	Set of nodes in the decomposed network, including the depot. The depot is denoted by 1, where the tour of the rebalancing vehicles start and end. Rest of the nodes are numbered from $2, 3, \ldots, \mathcal{N}_o$				
	$\mathcal{E}_o$	Edges in the decomposed network				
	$\bar{c}_{ij}$	Time taken to travel from node i to node j, $\forall (i,j) \in \mathcal{E}_o$ in seconds				
	$\bar{d}_i$	Imbalance at node i, $\forall i \in \mathcal{N}_o$. $\bar{d}_i = +1$ when node i is a pickup node or has a surplus of one bike, $\bar{d}_i = -1$ when node i is a delivery node or has a deficit of one bike and $\bar{d}_i = 0$ when node i is the depot				
	$\bar{\tau}$	Average loading unloading time per bike in seconds		✓		
	$\mathcal{V}$	Fleet of homogeneous rebalancing vehicles				
	$\mathcal{Q}$	Capacity of each vehicle in the fleet of homogeneous rebalancing vehicles				
Variables	τ_i	Arrival time of a rebalancing vehicle at node i, $\forall i \in \mathcal{N}_o$	×	✓		×
	x_{ij}	Equals 1, if Edge (i,j) is traversed by a rebalancing vehicle, otherwise it is 0, $\forall (i,j) \in \mathcal{E}_o$				
	q_{ij}	Quantity of bikes carried by a rebalancing vehicle on Edge (i,j), $\forall (i,j) \in \mathcal{E}_o$				
	$\mathcal{T}_v$	Tour of rebalancing vehicle v, $\forall v \in \mathcal{V}$		×		✓
	$\mathcal{T}$	$\{\mathcal{T}_v, \forall v \in \mathcal{V}\}$				
	$\mathcal{I}_v$	Loading unloading instructions of rebalancing vehicle v, $\forall v \in \mathcal{V}$				
	$\mathcal{I}$	$\{\mathcal{I}_v, \forall v \in \mathcal{V}\}$				
	k, l, m, n	Indices used for locations inside a Tour ($\mathcal{T}_v$) of a rebalancing vehicle				
	i_o, j_o	Iterators used in Algorithms 2, 3 and 6				

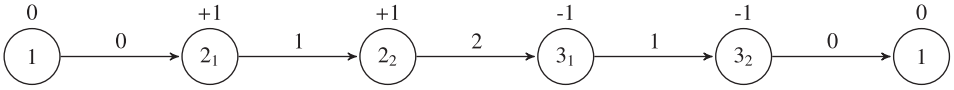

Fig. 5. Feasible solution for the decomposed network with a single vehicle of capacity 2.

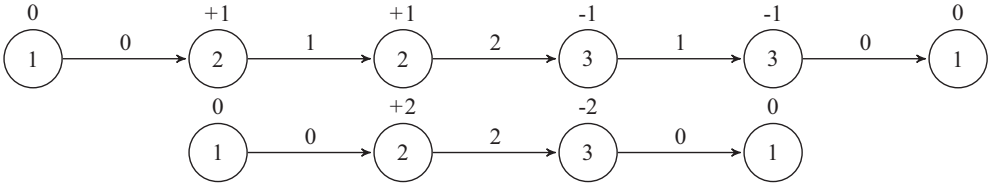

Fig. 6. Corresponding feasible solution for the original network with a single vehicle of capacity 2.

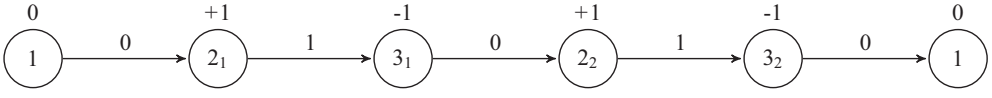

Fig. 7. Feasible solution for the decomposed network with a single vehicle of capacity 1.

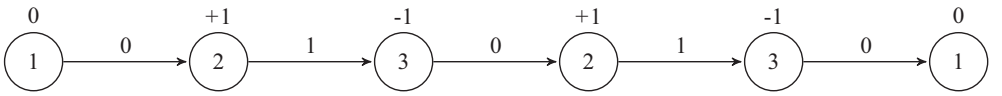

Fig. 8. Corresponding feasible solution for the original network with a single vehicle of capacity 1.

sponding feasible solution on the original network. This example illustrates how multiple visits to a node in the original network translates into single visit to a node in the decomposed network.

Before moving onto the Mixed Integer Linear Program (MILP) of SCRP on the decomposed network, we have to understand that in this article, we only address SCRP and not SPRP. For a particular instance to be feasible for SCRP, the total positive imbalance must equal the total negative imbalance, i.e., $\sum_{i\in\mathcal{N}} d_i = 0$ on the original network and by extension $\sum_{i\in\mathcal{N}_o} \overline{d}_i = 0$ on the decomposed network. From a modeling point of view, SCRP for SBBS is exactly similar to that for FFBS, i.e., same set of equalities and inequalities define the constraints. The only difference is from a computational point of view. In case of FFBS, the scale (number of nodes to rebalance and or number of bikes to rebalance) of SCRP can become very large compared to that of SBBS with a similar scale (total number of nodes and total number of bikes in the system).

(1)–(12) is the MILP of SCRP on the decomposed network. There are three decision variables in this formulation:

- τ_i is the arrival time of a rebalancing vehicle at node $i,\ \forall i \in \mathcal{N}_o$
- $x_{ij} = 1$, if Edge (i,j) is traversed by a rebalancing vehicle, otherwise it is 0, $\forall (i,j) \in \mathcal{E}_o$.
- q_{ij} is the quantity of bikes carried by a rebalancing vehicle on Edge $(i,j),\ \forall (i,j) \in \mathcal{E}_o$.

In this formulation, the objective function (1) is to minimize the make-span of the rebalancing fleet, i.e., τ_1. Constraints (2) and (3) are the Miller-Tucker-Zemlin Subtour Elimination constraints. Constraint (4) makes sure, that the number of visits to equals the number of exits from a node. Constraint (5) makes sure, that every node, other than the depot is visited exactly once. As there are $|\mathcal{V}|$ rebalancing vehicles and the flow is directional (because of capacity constraints), the number of uniques flows through the depot must be equal to $|\mathcal{V}|$, which is what Constraint (6) is. Each of the unique flow out of $|\mathcal{V}|$ flows, starting and ending at the depot corresponds to the tour for a rebalancing vehicle. Constraint (7) prevents self loops in the tour of rebalancing vehicles. Constraint (8) are the complete rebalancing constraints for each node in the graph. Constraint (8) ensures that the imbalance is met exactly at each node during rebalancing. Constraint (9) are the capacity constraints of the rebalancing vehicles.

$$\min\ \tau_1 \tag{1}$$

$$\text{s.t. } \tau_i \geqslant \tau_j + \overline{c}_{ji}x_{ji} + \overline{\tau} - M(1 - x_{ji}), \quad \forall i \in \mathcal{N}_o,\ j \in \mathcal{N}_o \setminus \{1\} \tag{2}$$

$$\tau_i \geqslant \overline{c}_{1i}x_{1i} - M(1 - x_{1i}), \quad \forall i \in \mathcal{N}_o \tag{3}$$

$$\sum_{j\in\mathcal{N}_o} x_{ji} = \sum_{j\in\mathcal{N}_o} x_{ij}, \quad \forall i \in \mathcal{N}_o \tag{4}$$

$$\sum_{j\in\mathcal{N}_o} x_{ji} = 1, \quad \forall i \in \mathcal{N}_o \setminus \{1\} \tag{5}$$

$$\sum_{j\in\mathcal{N}_o} x_{j1} = |\mathcal{V}| \tag{6}$$

$$\sum_{i\in\mathcal{N}_o} x_{ii} = 0 \tag{7}$$

$$\sum_{j\in\mathcal{N}_o} q_{ij} - \sum_{j\in\mathcal{N}_o} q_{ji} = \overline{d}_i, \quad \forall i \in \mathcal{N}_o \tag{8}$$

$$q_{ij} \leqslant \mathcal{Q}x_{ij}, \quad \forall (i,j) \in \mathcal{E}_o \tag{9}$$

$$\tau_i \geqslant 0, \quad \forall i \in \mathcal{N}_o \tag{10}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{E}_o \tag{11}$$

$$q_{ij} \in \mathcal{Z}^+, \quad \forall (i,j) \in \mathcal{E}_o \tag{12}$$

To illustrate how the above MILP can handle multiple vehicles, let us consider the network presented in Fig. 9. Fig. 10 presents a feasible solution on the network of Fig. 9 by a fleet consisting of 2 rebalancing vehicles with $\mathcal{Q} = 1$. Figs. 11 and 12 are the tours of Vehicle 1 and 2 respectively, for the feasible solution of Fig. 10 respectively. Further from Figs. 11 and 12, we can conclude that

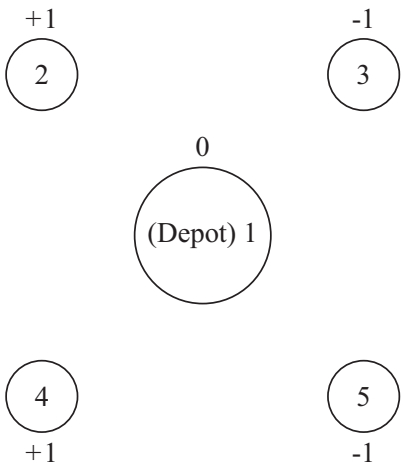

Fig. 9. Example network.

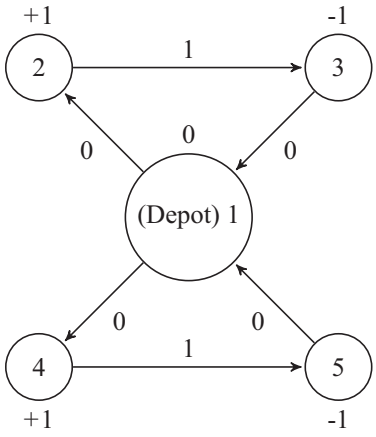

Fig. 10. A feasible solution of the network in Fig. 9 for a fleet consisting of 2 rebalancing vehicles.

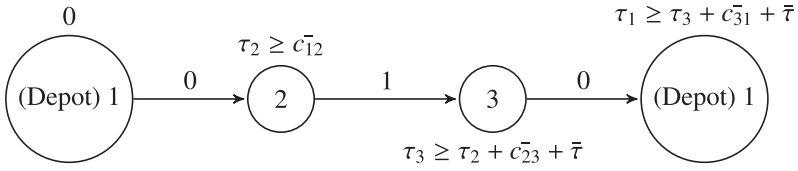

Fig. 11. Tour of vehicle 1 for the feasible solution in Fig. 10.

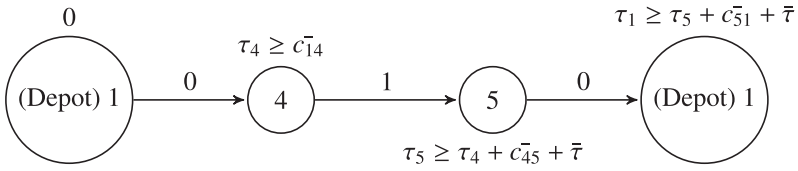

Fig. 12. Tour of vehicle 2 for the feasible solution in Fig. 10.

$$\tau_2 \geqslant \overline{c_{12}} \tag{13}$$
$$\tau_3 \geqslant \tau_2 + \overline{c_{23}} + \overline{\tau} \tag{14}$$
$$\tau_1 \geqslant \tau_3 + \overline{c_{31}} + \overline{\tau} \tag{15}$$
$$\tau_4 \geqslant \overline{c_{14}} \tag{16}$$
$$\tau_5 \geqslant \tau_4 + \overline{c_{45}} + \overline{\tau} \tag{17}$$
$$\tau_1 \geqslant \tau_5 + \overline{c_{51}} + \overline{\tau} \tag{18}$$

As our problem is a minimization problem, the above set of inequalities becomes equations as follows:

$$\tau_2 = \overline{c_{12}} \tag{19}$$
$$\tau_3 = \tau_2 + \overline{c_{23}} + \overline{\tau} \tag{20}$$
$$\tau_4 = \overline{c_{14}} \tag{21}$$
$$\tau_5 = \tau_4 + \overline{c_{45}} + \overline{\tau} \tag{22}$$

.

The above set of equations is pretty obvious except for the case of τ_1. If $\tau_3 + \overline{c_{31}} + \overline{\tau} > \tau_5 + \overline{c_{51}} + \overline{\tau}$, $\tau_1 = \tau_3 + \overline{c_{31}} + \overline{\tau}$, otherwise $\tau_1 = \tau_5 + \overline{c_{51}} + \overline{\tau}$. In case $\tau_3 + \overline{c_{31}} + \overline{\tau} = \tau_5 + \overline{c_{51}} + \overline{\tau}$, τ_1 can take either values. Thus τ_1 equals the arrival time of the rebalancing vehicle that arrives last at the depot.

The proposed formulation is computationally intractable even for small scale instances owing to the presence of Big M in the constraints and (geographic) symmetry in the decision variables, which make the linear programming relaxation of the formulation extremely weak. The Big M constraints are used for subtour elimination. Another reason for the computational intractability of the formulation is the significant increase in the number of variables in the formulation owing to spacial decomposition. It is directly proportional to square of the number of bikes to be rebalanced. Several strategies for strengthening and making the formulation computationally tractable is mentioned in Section 9.

4. Proposed heuristic

The algorithm proposed in this section is a hybrid of Nested Large Neighborhood Search and Variable Neighborhood Descent. It will be referred to as NLNS + VND in the rest of the article. NLNS + VND was greatly influenced by the success of metaheuristic algorithms based on perturbation and repair (Helsgaun, 2000, 2009; Ahuja et al., 2002; Applegate et al., 2003; Ghilas et al., 2016) for solving large scale traveling salesman and vehicle routing problems. NLNS + VND can solve SCRP on both the original and the decomposed network. There are major differences between NLNS + VND and previous algorithms reported in the literature (Chemla et al., 2013a; Erdoan et al., 2015) for solving SCRP in bike sharing systems. NLNS + VND consists of three primary components each of which have various sub-components. The three primary components are creating the initial solution, variable neighborhood descent for intensification and large neighborhood search for diversification. Each of these components and their corresponding sub-components are elaborated in great details in the subsequent sections. NLNS + VND has been coded in Julia (v 0.5.0) (Bezanson et al., 2012). The Julia implementation has been made as efficient as possible following recommendations from "http://docs.julialang.org/en/release-0.4/manual/performance-tips/" and "http://docs.julialang.org/en/release-0.4/manual/profile/", which include adding The "@inbounds" macro before each function definition. The sourcecode of the implementation is available at https://github.com/aritrasep/NLNS+VND.jl.

4.1. Initial solution

4.1.1. Single vehicle

For a fleet consisting of a single rebalancing vehicle, an initial solution is created using a greedy construction heuristic. Unlike Chemla et al. (2013a), both $(\mathcal{T}_v, \mathcal{I}_v)$ are created simultaneously. Algorithm 1 is the pseudo-code of the greedy construction heuristic. On lines 4 and 10, in Algorithm 1, the function **Maximum operations for every other node()** computes the maximum operation that can be performed at a node other than the current node, if that node is visited next from the current node. When computing maximum operation, only operations remaining at a node are taken into consideration. The procedure for computing Maximum Operations (or MaxOps) is very simple. First, we have to understand that $\text{sum}(I) =$ current number of bikes in the corresponding rebalancing vehicle. With this in mind, we can define $\text{MaxOps}_i = \begin{cases} \min\{d_i^o, Q - \text{sum}(I)\} & \text{if } d_i^o > 0 \\ \max\{d_i^o, -\text{sum}(I)\} & \text{if } d_i^o < 0 \\ 0 & \text{if } d_i^o = 0 \end{cases}$, $\forall i \in \mathcal{N}$. The **Nearest Neighbor Function()** in Algorithm 1 can be computed in three different ways. For all three different functions, only nodes with a non-zero maximum operation ($\text{MaxOps}_i \neq 0$), are considered for determining **Next node**.

- Nearest Neighbor 1: The nearest neighbor 1 of a node, is the node with the minimum (traveling) cost from the current node, i.e., $\arg\min(c_{ij}),\ \forall j \in \mathcal{N} \setminus \{i\};\ i$ being the current node.

- Nearest Neighbor 2: The nearest neighbor 2 of a node, is the node that has the maximum value of $\frac{|\text{Max Ops}_j|}{c_{ij}}$, $\forall j \in \mathcal{N} \setminus \{i\}$; i being the current node.
- Nearest Neighbor 3: The nearest neighbor 3 of a node, is a random node (where a non-zero operation is left) other than the current node and the depot.

From Line 5 in Algorithm 1, it is evident that the Next Node of the Depot is chosen randomly irrespective of what *Nearest Neighbor Function()* is used to generate the initial solution. This is because, in our experiments **Nearest Neighbor 2** is used to generate 10 different starting solutions. If the Next Node of the Depot is chosen according to *Nearest Neighbor 2*, all the 10 starting solutions will exactly be the same.

Algorithm 1. Greedy Construction Heuristic.

Data: d_i, Q, c_{ij}, Nearest Neighbor Function
Result: $(\mathcal{T}_v, \mathcal{I}_v)$
1 $\mathcal{T}_v \longleftarrow$ [Depot]
2 $\mathcal{I}_v \longleftarrow$ [0]
3 $d_i^o \longleftarrow d_i$
4 Max Ops $\longleftarrow$ Maximum operations for every other node($d_i^o, Q, sum(\mathcal{I}_v)$)
5 Next node $\longleftarrow$ Randomly select any node from 2 **to** $|\mathcal{N}|$
6 Add Next node at the end of $\mathcal{T}_v$
7 Add Max $\text{Ops}_{\text{Next node}}$ at the end of $\mathcal{I}_v$
8 $d^o_{\text{Next node}} = d^o_{\text{Next node}} - \text{Max Ops}_{\text{Next node}}$
9 **while** *Number of non zero elements in* $d_i^o > 0$ **do**
10 Max Ops $\longleftarrow$ Maximum operations for every other node($d_i^o, Q, sum(\mathcal{I}_v)$)
11 Next node $\longleftarrow$ Nearest Neighbor Function(d_i^o, c_{ij}, Next node, Max Ops)
12 Add Next node at the end of $\mathcal{T}_v$
13 Add Max $\text{Ops}_{\text{Next node}}$ at the end of $\mathcal{I}_v$
14 $d^o_{\text{Next node}} = d^o_{\text{Next node}} - \text{Max Ops}_{\text{Next node}}$
15 **end**
16 Add Depot at the end of $\mathcal{T}_v$
17 Add 0 at the end of $\mathcal{I}_v$

4.1.2. Multiple vehicles

When $|\mathcal{V}| > 1$, a partition first rebalance second approach is used. For partitioning, two strategies have been tried, first partitioning based on geographic locations of the nodes and second partitioning randomly. During our experiments, it was observed that, on average partitioning randomly is not only faster but also results in higher quality solutions for large scale instances. Thus, partitions are created randomly. However, the partition created in this stage is by no means the final partition, because the overall solution may get stuck in a local optima. To address this, local search operators **INTER Crossover, INTER Node Swapping, INTER Edge Exchange** have been proposed, to improve the quality of the solution by modifying the partitions as needed.

Exactly $|\mathcal{V}|$ partitions are made from the set of nodes $\mathcal{N}$, such that for each partition, the total deficit of bikes equals the total surplus of bikes. While partitioning, a node $\in \mathcal{N}$ may be split into 2 to up to $|\mathcal{V}|$ nodes, each of them being in a separate partition, however the sum of their absolute imbalance in each partition, must equal their total original absolute imbalance. This is necessary to satisfy the condition that, total deficit of bikes equals total surplus of bikes in each partition. Once $|\mathcal{V}|$ partitions have been created, Algorithm 1 is used to create an initial solution for each partition.

4.2. Variable Neighborhood Descent (VND)

In VND, the feasibility of the solution is maintained during the subsequent local search operations, i.e., no capacity constraints are violated. To keep a solution feasible in the following operations, when an operation is performed on T_v, the corresponding operation is also performed on I_v and vice versa. This has significant advantages, as it results in a decrease in the size of the neighborhood of an incumbent solution to a great extent, taking considerably less time to explore them. Further, a candidate list based on nearest neighbors is used to rank the edges in a tour. Only operations that result in a tour whose highest edge rank is less than or equal to the highest edge rank of the current tour are allowed. This prevents exploration of unwanted regions in a neighborhood and subsequently reduces exploration time, making the operation extremely granular (Toth and Vigo, 2003). In total, seven such granular local search operators have been developed and used successively inside Variable Neighborhood Descent (VND). Operations which have an **INTRA** or an **INTER** prefix signifies that the

operation takes place inside a solution of a single vehicle or the operation takes place between solutions of two vehicles respectively. Further a **K-OPT Operation** means that **K** variables are altered from their present state, in this case **K** edges are changed. The higher the value of **K**, the more is the likely improvement, but it comes at the cost of computing time. For a detailed understanding and analysis of **K-OPT Operations** the readers are referred to Helsgaun (2000, 2009).

4.2.1. Local search operators for single vehicle

INTRA Delete-Reinsert: In this case, one node is deleted from a location and inserted in another location in $\mathcal{T}_v$ while maintaining feasibility of the solution. At each iteration, starting from the left (or beginning) of $\mathcal{T}_v$, each node (other than the depot) is inspected to determine if it can be deleted and reinserted either ahead or behind its current location in $\mathcal{T}_v$. If a valid location(s) is (are) found that reduces the cost of $\mathcal{T}_v$, then the (best) operation is made. An iteration is completed on reaching the penultimate location in $\mathcal{T}_v$. This procedure is continued until no further improvement is possible.

INTRA Node Swapping: Two 4-OPT neighborhoods based on Intra Node Swapping have been developed. Let k & l be locations of two nodes in $\mathcal{T}_v$, such that $k \neq l,\ k < l,\ k \neq 1$ & $l \neq |\mathcal{T}_v|$. If $\mathcal{I}_{v_k} = \mathcal{I}_{v_l}$ and $\mathcal{T}_{v_k} \neq \mathcal{T}_{v_l}$, then the two locations k and l can be swapped to obtain a new feasible tour. If the swapping results in a tour with a lower cost, the swapping is confirmed. At each iteration, each location (k) of $\mathcal{T}_v$ is inspected for a possible swap. If a valid swap(s) is(are) found that reduces the current cost of $\mathcal{T}_v$, the (best) swap is made. An iteration is completed on reaching the penultimate location in $\mathcal{T}_v$. This procedure is continued until no further improvement is possible. Another variation of the above procedure is, let k and l be locations of two nodes in $\mathcal{T}_v$, such that $k \neq l,\ k < l,\ k \neq 1$ and $l \neq |\mathcal{T}_v|$. If $\mathcal{I}_{v_k} \neq \mathcal{I}_{v_l}$ and $\mathcal{T}_{v_k} \neq \mathcal{T}_{v_l}$, and swapping the two nodes at k and l results in a new feasible tour with a lower cost, the swapping is confirmed. Everything else is exactly same as the above procedure.

INTRA Edge Exchange: Let us consider four edges e_k, e_l, e_m, e_n, connecting nodes at location k, l, m, n to nodes at location $k+1,\ l+1,\ m+1,\ n+1$ in $\mathcal{T}_v$ respectively, such that $k < l < m < n$. If the flow of bikes (= number of bikes carried by the rebalancing vehicle) on edges e_k and e_l and that on edges e_m & e_n are equal, the tour segment $(k+1) \rightarrow l$ can be swapped with the tour segment $(m+1) \rightarrow n$ without violating any capacity constraints. If such a operation results in a tour with a lower cost than the current one, that operation is confirmed. At each iteration, a step ($= l - k - 1$) is fixed and all possible values of m and n are checked for possible exchanges. If no improvement is possible for the current value of step, its value is incremented by 1. The value of step is initialized with 1 and can at most be increased to $|\mathcal{T}_v| - 3$, otherwise there will be overlapping of the tour segments. On reaching this value of step, the operation is terminated.

INTRA Adjust Instructions: In this operation, number of edges broken, which is variable equals number of edges inserted. The number of edges broken or inserted, can be represented as $2n$, where $n \in Z^+$. This operation comprises of nodes, that are visited multiple times in the current tour. The objective of this neighborhood is to drive the loading-unloading instructions, at viable locations in the current tour, of such nodes towards 0. If this can be achieved for one or more locations, the corresponding locations can be removed from the tour. This is because, the instances (used in this study) are metric, and removing nodes at locations in the current tour, for which the instruction is 0, reduces the cost of the tour without making it infeasible. Let a node be present at the kth and lth, $k < l$ locations in $\mathcal{T}_v$. If $|I_{v_k}| \leqslant |I_{v_l}|,\ I_{v_k}$ is driven towards 0 by transferring instructions between I_{v_k} and I_{v_l}, while maintaining feasibility of the flow of bikes in the tour segment $k \longrightarrow l$ and vice versa. This procedure is executed for all possible combinations of k and l for each node with multiple visits in $\mathcal{T}_v$. This operation also serves a secondary purpose. Instructions at some locations in the current tour are altered, which might create new operations for other local search operators described earlier.

4.2.2. Local search operators for multiple vehicles

INTER Crossover: This is a 2-OPT operation in which part of the tour of a vehicle is swapped with part of the tour of another vehicle while maintaining feasibility of the solution. At each iteration, starting from the left (or beginning) of $\mathcal{T}_{v1}$ (tour of first vehicle), flow of bikes on each edge is inspected to determine if it can be swapped with an edge on $\mathcal{T}_{v2}$ (tour of another vehicle) with exactly same flow of bikes. If a valid edge(s) is (are) found that reduces the current make-span, then the (best) operation is made. An iteration is completed on reaching the penultimate location in $\mathcal{T}_{v1}$. This procedure is continued until no further improvement is possible.

INTER Node Swapping: Let k and l be location of two nodes in $\mathcal{T}_{v1}$ and $\mathcal{T}_{v2}$ respectively. If $\mathcal{I}_{v1_k} = \mathcal{I}_{v2_l}$ and $\mathcal{T}_{v1_k} \neq \mathcal{T}_{v2_l}$, then the two locations k and l can be swapped without violating any capacity constraints. If the swapping results in a decrease in the current make-span, the swapping is confirmed. At each iteration, each location (k) of $\mathcal{T}_{v1}$ is inspected for a possible swap. If a valid swap(s) is(are) found that reduces the current cost of the tour, the (best) swap is made. An iteration is completed on reaching the penultimate location in $\mathcal{T}_{v1}$. This procedure is continued until no further improvement is possible.

INTER Edge Exchange: Let us consider four edges e_k, e_l, e_m, e_n, connecting nodes at location k, l, m, n to nodes at location $k+1,\ l+1,\ m+1,\ n+1$ in $\mathcal{T}_{v1}$ and $\mathcal{T}_{v2}$ respectively, such that $k < l$ and $m < n$. If the flow of bikes (= number of bikes carried by the rebalancing vehicle) on edges e_k and e_l and that on edges e_m and e_n are equal, the tour segment $(k+1) \rightarrow l$ in $\mathcal{T}_{v1}$ can be swapped with the tour segment $(m+1) \rightarrow n$ in $\mathcal{T}_{v2}$ without violating any capacity constraints. If such a operation results in a decrease of the make-span, that operation is confirmed. At each iteration, a step ($= l - k - 1$) is fixed and all possible values of m and n are checked for possible exchanges. If no improvement is possible for the current value

of step, its value is incremented by 1. The value of step is initialized with 1 and can at most be increased to length of the current tour – 5, otherwise there will be overlapping of the tour segments. On reaching this value of step, the procedure is terminated.

Algorithm 2 is the pseudo-code of VND, used in NLNS + VND. In VND, local search operations (described in the above sub-sections) are done sequentially in an iterative manner on an incumbent solution, until no further improvement is possible. The order in which the operations is carried out is a crucial factor for the improvement to be substantial. In VND, it is based on the computational complexity and the execution time of an individual operation. The order used in VND is as follows:

- INTRA Adjust Instructions (Operation_1)
- INTRA Delete-Reinsert (Operation_2)
- INTRA Node Swap (Operation_3)
- INTRA Edge Exchange (Operation_4)
- INTER Crossover (Operation_5)
- INTER Node Swap (Operation_6)
- INTER Edge Exchange (Operation_7)

Algorithm 2. Variable Neighborhood Descent.

Data: $\mathcal{T}, \mathcal{I}, \mathrm{d_i}, Q, \mathrm{c_{ij}}, \bar{\tau}$
Result: $(\mathcal{T}, \mathcal{I})$
STOP $\longleftarrow$ FALSE
while *STOP = FALSE* **do**
  for $v \leftarrow 1$ **to** $|\mathcal{V}|$ **do**
    while $Cost(\mathcal{T}_v) \neq Cost(\mathcal{T}_v^{previous})$ **do**
      for $i_o \leftarrow 1$ **to** 4 **do**
        $(\bar{\mathcal{T}}_v, \bar{\mathcal{I}}_v) \longleftarrow \text{Operation}_{i_o}(\mathcal{T}_v, \mathcal{I}_v, \mathrm{d_i}, Q, \mathrm{c_{ij}})$
        if $Cost(\bar{\mathcal{T}}_v) < Cost(\mathcal{T}_v)$ **then**
          $\mathcal{T}_v \longleftarrow \bar{\mathcal{T}}_v$
          $\mathcal{I}_v \longleftarrow \bar{\mathcal{I}}_v$
        end
      end
    end
    STOP $\longleftarrow$ TRUE
  end
  if $|\mathcal{V}| > 1$ **then**
    while $Makespan(\mathcal{T}) \neq Makespan(\mathcal{T}^{previous})$ **do**
      for $i_o \leftarrow 5$ **to** 7 **do**
        $(\bar{\mathcal{T}}, \bar{\mathcal{I}}) \longleftarrow \text{Operation}_{i_o}(\mathcal{T}, \mathcal{I}, \mathrm{d_i}, Q, \mathrm{c_{ij}}, \bar{\tau})$
        if $Cost(\bar{\mathcal{T}}) < Cost(\mathcal{T})$ **then**
          $\mathcal{T} \longleftarrow \bar{\mathcal{T}}$
          $\mathcal{I} \longleftarrow \bar{\mathcal{I}}$
        end
      end
    end
    if $Makespan(\mathcal{T}) \neq Makespan(\mathcal{T}^{previous})$ **then**
      STOP $\longleftarrow$ FALSE
    end
  end
end

4.3. Large Neighborhood Search (LNS)

With the decrease in the size of the neighborhood used in VND, finding high-quality solutions becomes extremely challenging. To overcome this, VND is hybridized with Large Neighborhood Search (LNS). Multiple large neighborhoods are explored to find local optimas that, either are clustered together or are present in valleys far away from each other. Further,

these large neighborhoods are nested together to increase the effectiveness of LNS. The perturbation and repairing operators that comprise LNS are described in the subsequent sections. They are only applicable for single vehicles. In case the fleet comprises of multiple vehicles, it is applied to each partition corresponding to each vehicle. Thus in Algorithms 3–5, when $|\mathcal{V}| > 1$, d_i represent the imbalance of the nodes for the partition, where the corresponding vehicle is performing the rebalancing.

4.3.1. Repairing operators

The repairing operator proposed in this section is capable of repairing a partial or an infeasible solution of a single vehicle. Algorithm 3 comprises of multiple vehicles, is the pseudo-code of the repairing operator. It is based on the greedy construction heuristic described in Section 4.1. As, three different functions (Nearest Neighbor 1, 2 and 3 described in Section 4.1.1) can be used as the nearest neighbor function (line 19), often three distinct solutions can be constructed from an initial partial solution. This feature comes in handy while repairing an infeasible solution from a perturbation (Section 4.3.2) on a feasible solution.

Algorithm 3. Repair Tour.

Data: $\mathcal{T}_v$, $\mathrm{d_i}$, Q, $\mathrm{c_{ij}}$, Nearest Neighbor Function
Result: $(\mathcal{T}_v, \mathcal{I}_v)$
1 **if** $\mathcal{T}_v[1]$ *is not Depot* **then**
2 Depot is added at the beginning of $\mathcal{T}_v$
3 **end**
4 $\mathcal{I}_v \longleftarrow [0]$
5 $\mathrm{d_i^o} \longleftarrow \mathrm{d}_i$
6 $k \longleftarrow 2$
7 **while** $k \leq |\mathcal{T}_v|$ **do**
8 Max Ops $\longleftarrow$ Maximum operations for every other node($\mathrm{d_i^o}, Q, sum(\mathcal{I}_v)$)
9 **if** $Max\ Ops_{\mathcal{T}_v[k]} = 0$ **then**
10 Delete node at location k of $\mathcal{T}_v$
11 **else**
12 Add Max $\mathrm{Ops}_{\mathcal{T}_v[k]}$ at the end of $\mathcal{I}_v$
13 $\mathrm{d^o_{\mathcal{T}_v[k]}} \longleftarrow \mathrm{d^o_{\mathcal{T}_v[k]}} - \mathrm{Max\ Ops_{\mathcal{T}_v[k]}}$
14 $k \longleftarrow k + 1$
15 **end**
16 **end**
17 **while** *Number of non zero elements in* $d_i^o > 0$ **do**
18 Max Ops $\longleftarrow$ Maximum operations for every other node($\mathrm{d_i^o}, Q, sum(\mathcal{I}_v)$)
19 Next node $\longleftarrow$ Nearest Neighbor Function($\mathrm{d_i^o}, \mathrm{c_{ij}}$, Next node, Max Ops)
20 Add Next node at the end of $\mathcal{T}_v$
21 Add Max $\mathrm{Ops_{Next\ node}}$ at the end of $\mathcal{I}_v$
22 $\mathrm{d^o_{Next\ node}} = \mathrm{d^o_{Next\ node}} - \mathrm{Max\ Ops_{Next\ node}}$
23 **end**
24 Add Depot at the end of $\mathcal{T}_v$
25 Add 0 at the end of $\mathcal{I}_v$

4.3.2. Perturbation operators

Perturbation operators used in NLNS + VND are greatly influenced by Chained Lin-Kernighan used for solving large-scale Traveling Salesman Problems (Applegate et al., 2003). However, a major difference of the two methods is that in Chained Lin-Kernighan selected edges are destroyed whereas in the proposed perturbation operators selected locations in a tour are destroyed. Locations in a tour are ranked for the purpose of perturbation.

$$\text{Rank of location k} = \frac{\text{Rank of Edge}_{k-1,k} + \text{Rank of Edge}_{k,k+1}}{2}, \quad \forall k \in [2, \text{length of tour} - 1]$$

and

$$\text{Average Rank of a Tour} = \frac{\sum_{k\in[2,\text{length of tour}-1]} \text{Rank of location k}}{\text{length of tour} - 2}$$

The higher the rank the less desirable the location in the tour and more likelihood of it being destroyed during a perturbation.

In the upcoming perturbation operators, two functions **Sorted Location Rank List** and **Reverse Sorted Location Rank List** are used extensively. **Sorted Location Rank List** takes a tour and a rank list as its inputs and computes the rank of all the locations (except for the first and for the last) in the tour. It then sorts the locations in the tour in a descending order of their respective ranks. The sorted list and the number of locations whose rank is above the average rank of the tour is returned. **Reverse Sorted Location Rank List** is similar to **Sorted Location Rank List**, except that the locations in the tour are sorted in an ascending order of their respective ranks. The sorted list and the number of locations whose rank is less than or equal to the average rank of the tour is returned.

4.3.2.1. Perturbation operators 1 and 2. Perturbation operators 1 and 2 are complements of each other. In Perturbation Operator 1, local optimas in valleys clustered together, are explored systematically by destroying locations in a tour with undesirable configurations, followed by repairing of the tour and VND. If the cost of the new tour is lower than that of the current tour, the new tour becomes the current tour, otherwise the value of perturbation is incremented. The process continues until the value of perturbation equals that of maximum perturbation. Similarly, in Perturbation operator 2, local optimas in valleys far away from each other, are explored systematically by destroying locations in a tour with undesirable configurations, followed by repairing of the tour and VND. If the cost of the new tour is lower than that of the current tour, the new tour becomes the current tour, otherwise the value of perturbation is incremented. The process continues until the value of perturbation equals that of maximum perturbation.

With increase in $|\mathcal{N}|$, execution time of VND and the **maximum perturbation** increases significantly. Thus, to keep the exploration time of these large neighborhoods reasonable, without hampering the quality of the solutions found, number of perturbations is limited to only $\frac{15}{|\mathcal{N}|}\%$ of total perturbations possible. Further, the value of perturbation is varied between its minimum and maximum values simultaneously. This is based on the observation that, perturbation is most effective when it is close to its extreme values.

Algorithm 4 is the pseudo-code for Perturbation Operator 1. The pseudo-code for Perturbation Operator 2 is similar to Algorithm 4, except **Sorted Location Rank List** and **Number of Locations above Average Rank** are replaced by **Reverse Sorted Location Rank List** and **Number of Locations below Average Rank** respectively.

Algorithm 4. Perturbation Operator 1.

Data: $\mathcal{T}_v, \mathcal{I}_v, \mathrm{d_i}, Q, \mathrm{c_{ij}}$,Repairing Operator
Result: $(\mathcal{T}_v, \mathcal{I}_v)$
Location Rank List,Number of Locations above Average Rank = Sorted Location Rank List($\mathcal{T}_v$,Rank List)
Perturbation $\longleftarrow$ 1
while *Perturbation* $\leq \dfrac{\textit{Number of Locations above Average Rank} \times 15}{|\mathcal{N}|}$ **do**
  if *Perturbation is Odd* **then**
    Locations to Destroy $\longleftarrow$ Location Rank List $\left[1\textbf{to}\dfrac{\text{Perturbation}+1}{2}\right]$
  else
    Locations to Destroy $\longleftarrow$ Location Rank List$\left[1\textbf{to}\text{Number of Locations above Average Rank} + 1 - \dfrac{\text{Perturbation}}{2}\right]$
  end
  $\bar{\mathcal{T}}_v \longleftarrow$ Delete Nodes at locations [Locations to Destroy] of $\mathcal{T}_v$
  $(\bar{\mathcal{T}}_v, \bar{\mathcal{I}}_v) \longleftarrow$ Repair Tour($\bar{\mathcal{T}}_v, \mathrm{d_i}, Q, \mathrm{c_{ij}}$,Repairing Operator)
  $(\bar{\mathcal{T}}_v, \bar{\mathcal{I}}_v) \longleftarrow$ VND($\bar{\mathcal{T}}_v, \bar{\mathcal{I}}_v, \mathrm{d_i}, Q, \mathrm{c_{ij}}$)
  if *Cost*($\bar{\mathcal{T}}_v$) < *Cost*($\mathcal{T}_v$) **then**
    $\mathcal{T}_v \longleftarrow \bar{\mathcal{T}}_v$
    $\mathcal{I}_v \longleftarrow \bar{\mathcal{I}}_v$
    Location Rank List,Number of Locations above Average Rank = Sorted Location Rank List($\mathcal{T}_v$,Rank List)
    Perturbation $\longleftarrow$ 1
  else
    Perturbation $\longleftarrow$ Perturbation+1
  end
end

4.3.2.2. Perturbation Operators 3 and 4. As with Perturbation Operators 1 and 2, Perturbation Operators 3 and 4 are also complements of each other. In Perturbation Operator 3, node(s) on the right of a location in the tour is (are) destroyed, followed by repairing of the tour and VND. If the cost of the new tour is lower than that of the current tour, the new tour becomes the current tour, otherwise the value of perturbation is incremented. The process continues until the value of perturbation equals that of maximum perturbation. Similarly, in Perturbation Operator 4, node(s) on the left of a location in the tour is (are) destroyed, followed by repairing of the tour and VND. If the cost of the new tour is lower than that of the current tour, the new tour becomes the current tour, otherwise the value of perturbation is incremented. The process continues until the value of perturbation equals that of maximum perturbation. The locations chosen in these large neighborhoods are locations with rank greater than the average rank of the tour.

Algorithm 5 is the pseudo-code for Perturbation Operator 3. Pseudo-code for Perturbation Operator 4 is similar to Algorithm 5, except $>$ in lines 5 and 23 and **Delete nodes in $\mathcal{T}_v$ right of (Location to Destroy[Perturbation])** in line 13 is replaced by $<$ and **Delete nodes in $\mathcal{T}_v$ left of (Location to Destroy[Perturbation])** respectively.

Algorithm 5. Perturbation Operator 3.

```
   Data: T_v, I_v, d_i, Q, c_ij, Repairing Operator
   Result: (T_v, I_v)
 1 Location Rank List,Number of Locations above Average Rank = Sorted Location Rank List(T_v,Rank List)
 2 Location Rank List ⟵ Location Rank List[1 to Number of Locations above Average Rank]
 3 k ⟵ 2
 4 while k ≤ —Location Rank List—) do
 5     if Location Rank List[k] > Location Rank List[k − 1] then
 6         delete at(Location Rank List,k)
 7     else
 8         k ⟵ k + 1
 9     end
10 end
11 Perturbation ⟵ 1
12 while Perturbation ≤ —Location Rank List— do
13     T̄_v ⟵ Delete Nodes in T_v right of (Location to Destroy[Perturbation])
14     (T̄_v, Ī_v) ⟵ Repair Tour(T̄_v, d_i, Q, c_ij, Repairing Operator)
15     (T̄_v, Ī_v) ⟵ VND(T̄_v, Ī_v, d_i, Q, c_ij)
16     if Cost(T̄_v) < Cost(T_v) then
17         T_v ⟵ T̄_v
18         I_v ⟵ Ī_v
19         Location Rank List,Number of Locations above Average Rank = Sorted Location Rank List(T_v,Rank
             List)
20         Location Rank List ⟵ Location Rank List[1 to Number of Locations above Average Rank]
21         k ⟵ 2
22         while k ≤ —Location Rank List— do
23             if Location Rank List[k] ¿ Location Rank List[k − 1] then
24                 delete at(Location Rank List,k)
25             else
26                 k ⟵ k + 1
27             end
28         end
29         Perturbation ⟵ 1
30     else
31         Perturbation ⟵ Perturbation+1
32     end
33 end
```

4.4. NLNS + VND

Algorithm 6 is the pseudo-code of NLNS + VND. The initial solution is constructed using Algorithm 1 (line 2). *Repairing Operator*$_{j_o}$ in line 9 in Algorithm 6, denotes that *Nearest Neighbor* $j_o()$ is used in Algorithm 3 for repairing the perturbed solution. $\mathcal{T}_v^{-12}$ is the 12th previous tour after perturbation.

Algorithm 6. Nested Large Neighborhood Search + Variable Neighborhood Descent – NLNS + VND.

Data: $d_i, Q, c_{ij}, \bar{\tau}$
Result: $(\mathcal{T}, \mathcal{I})$
for $v \leftarrow 1$ **to** $-\mathcal{V}-$ **do**
  $(\mathcal{T}_v, \mathcal{I}_v) \longleftarrow$ Greedy Construction Heuristic(d_i, Q, c_{ij},Nearest Neighbor 2)
end
STOP $\longleftarrow$ FALSE
while *STOP = FALSE* **do**
  for $v \leftarrow 1$ **to** $-\mathcal{V}-$ **do**
    for $i_o \leftarrow 1$ **to** 4 **do**
      for $j_o \leftarrow 1$ **to** 3 **do**
        $(\bar{\mathcal{T}}_v, \bar{\mathcal{I}}_v) \longleftarrow$ Perturbation Operator $i_o(\mathcal{T}_v, \mathcal{I}_v, d_i, Q, c_{ij}$,Repairing Operator $j_o)$
        if $Cost(\bar{\mathcal{T}}_v) < Cost(\mathcal{T}_v)$ **then**
          $\mathcal{T}_v \longleftarrow \bar{\mathcal{T}}_v$
          $\mathcal{I}_v \longleftarrow \bar{\mathcal{I}}_v$
        end
        if $Cost(\bar{\mathcal{T}}_v) = Cost(\mathcal{T}_v^{-12})$ **then**
          STOP $\longleftarrow$ TRUE
          break
        end
      end
      if *STOP = TRUE* **then**
        break
      end
    end
  end
  if $|\mathcal{V}| > 1$ **then**
    $(\bar{\mathcal{T}}, \bar{\mathcal{I}}) \longleftarrow$ VND($\mathcal{T}, \mathcal{I}, d_i, Q, c_{ij}, \bar{\tau}$)
    if $Makespan(\bar{\mathcal{T}}) < Make\text{-}span(\mathcal{T})$ **then**
      $\mathcal{T} \longleftarrow \bar{\mathcal{T}}$
      $\mathcal{I} \longleftarrow \bar{\mathcal{I}}$
      STOP $\longleftarrow$ FALSE
    end
  end
end

5. Recommended solution strategies

Instances of SCRP can be classified into two categories:

1. Instances with zero imbalance at the Depot, i.e., $d_1 = 0$,
2. Instances with non-zero imbalance at the Depot, i.e., $d_1 \neq 0$,

The proposed MILP and NLNS + VND can only handle instances with zero imbalance at the depot. In this section, it is shown how an instance with non-zero imbalance at the depot can be converted into an instance with zero imbalance at the depot, so that the proposed methodology becomes applicable.

5.1. Instances with zero imbalance at the Depot

The recommended method is using NLNS + VND to solve the instance on the original network. Currently, the proposed MILP is computationally intractable. However, if using some techniques the MILP can be made computationally tractable, it can be used to solve the instance in the following way:

1. Decompose the original network of the instance into its decomposed network
2. Solve the MILP on the decomposed network
3. Convert the solution found by the MILP on the decomposed network to a solution on the original network

5.2. Instances with non-zero imbalance at the Depot

For instances, where the depot has a non-zero imbalance, i.e., $d_1 \neq 0$, a pseudo node is created in the network. The modified network has $|\mathcal{N}+1|$ nodes, $|\mathcal{N}+1|$ being the index of the pseudo node created. In the modified network,

$$\tilde{d}_1 = 0 \tag{23}$$
$$\tilde{d}_{|\mathcal{N}+1|} = d_1 \tag{24}$$
$$\tilde{d}_i = d_i, \quad \forall i \in \mathcal{N} \setminus \{1\} \tag{25}$$
$$\tilde{c}_{ij} = c_{ij}, \quad \forall i \in \{1, \ldots, |\mathcal{N}+1|\},\ j \in \mathcal{N} \tag{26}$$
$$\tilde{c}_{i|\mathcal{N}+1|} = c_{i1}, \quad \forall i \in \{1, \ldots, |\mathcal{N}+1|\} \tag{27}$$
$$\tilde{c}_{|\mathcal{N}+1|i} = c_{1i}, \quad \forall i \in \{1, \ldots, |\mathcal{N}+1|\} \tag{28}$$

For a feasible solution on the modified network, all $|\mathcal{N}+1|$ in the tour must be replaced by **1** to obtain a feasible solution on the original network. The recommended method in this scenario is the following:

1. When NLNS + VND is used to solve the SCRP:
    1. Convert the original network of the instance into its modified original network, by adding a pseudo node and making the imbalance of the depot equal to zero as mentioned above.
    2. Solve SCRP on the modified original network using NLNS + VND
    3. Change index of the pseudo node in the solution obtained by NLNS + VND to **1** so that it becomes a feasible solution for the original network.
2. When MILP is used to formulate and solve the SCRP:
    1. Convert the original network of the instance into its modified original network, by adding a pseudo node and making the imbalance of the depot equal to zero as mentioned above.
    2. Decompose the modified original network of the instance into its decomposed network
    3. Solve the MILP on the decomposed network
    4. Convert the solution obtained by the MILP on the decomposed network to a solution on the modified original network
    5. Change index of the pseudo node in the solution for the modified original network to **1** so that it is a feasible solution for the original network.

6. Case Study 1: 1-PDTSP instances

The purpose of Case Study 1 is to compare the performance of **NLNS + VND** with exact algorithms from Erdoan et al. (2015) and Tabu Search algorithms from Chemla et al. (2013a).

The instances are those used in Chemla et al. (2013a) and Erdoan et al. (2015) and adapted from the 1-PDTSP instances introduced in Hernndez-Prez and Salazar-Gonzlez (2004). The instances are available at https://github.com/aritrasep/BSSLib.jl. Computational experiments are carried out on instances with $\alpha = \{1, 3\}$, $|\mathcal{N}| = \{20, 30, 40, 50, 60\}$, $Q = \{10, 15, 20, 25, 30, 35, 40, 45, 1000\}$ and $|\mathcal{N}| = 100$, $Q = \{10, 30, 45, 1000\}$. Further, each of these configurations have 10 independent instances, resulting in a total of 980 instances. For each instance, we compute $d_i = \text{Original Imbalance} \times \alpha,\ \forall i \in \mathcal{N}$ and c_{ij} as the euclidean distance between the nodes i and j, $\forall i, j \in \mathcal{N}$ (see Tables 3 and 4).

For comparing NLNS + VND with Exact (Erdoan et al., 2015) as well as Tabu Search (Chemla et al., 2013a) Algorithms, the test cases are divided into 3 sets as described in Table 5. $\alpha \in \{1, 3\}$ for all the instances in all 3 sets.

The best known lower bound reported in the literature as mentioned in Table 6, is the maximum of the lower bounds reported by Erdoan et al. (2015) and Chemla et al. (2013a) for each instance. Negative values in column Gap under PEA in Table 7 denotes the added value of preemption.

Table 3
Terminology of the algorithms.

Terminology	Description
TS1	Tabu search algorithm initialized with solution from greedy heuristic (Chemla et al., 2013a)
TS2	Tabu search algorithm initialized with Eulerian circuit from MILP relaxation (Chemla et al., 2013a)
RB	Reliability branching used for exploring the Branch-and-bound tree while solving relaxation of the static rebalancing problem (Chemla et al., 2013a)
DB	Degree branching used for exploring the Branch-and-bound tree while solving relaxation of the static rebalancing problem (Chemla et al., 2013a)
PEA	Exact algorithm that allows preemption (Erdoan et al., 2015)
NPEA	Exact algorithm that does not allow preemption (Erdoan et al., 2015)
NLNS + VND	Hybrid nested large neighborhood search with variable neighborhood descent algorithm presented in this paper

Table 4
Number of trials of each algorithm.

Algorithm	Number of trials
TS1	2
TS2	1 trial each, with RB and DB as branching strategy
NPEA	1
PEA	1
NLNS + VND	10 (each with Nearest Neighbor 2 as initial solution creator)

Table 5
Configuration of each set.

Set	Configuration		Comparing NLNS + VND with	
	$\mathcal{N}$	Q	Exact algorithms	Tabu search algorithms
I	20, 40, 60	10, 30, 45, 1000	✓	✓
II	20, 30, 40, 50, 60	10, 15, 20, 25, 30, 35, 40, 45, 1000	✓	✗
III	20, 40, 60, 100	10, 30, 45, 1000	✗	✓

Table 6
Description of parameters used in Table 7.

Type of measure	Algorithms	Description	Unit
Gap	Exact	Absolute Gap of Upperbound found by Exact Algorithms from best known Lowerbound reported in literature	%
Best Gap	Tabu Search and NLNS + VND	Absolute Gap of Best Upperbound found by corresponding Algorithms in all of its trials, from best known Lowerbound reported in literature	%
Avg Gap	Tabu Search and NLNS + VND	Absolute Gap of Average Upperbound found by corresponding Algorithms from all of its trials, from best known Lowerbound reported, in literature	%
Time	Exact, Tabu Search and NLNS + VND	Computation time of corresponding algorithms standardized to our workstation	sec

Table 7
Summary of overall results for 1-PDTSP instances.

Set	Exact Algorithms				Tabu Search Algorithms						NLNS + VND		
	PEA		NPEA		TS1			TS2					
	Gap	Time	Gap	Time	Best Gap	Avg Gap	Time	Best Gap	Avg Gap	Time	Best Gap	Avg Gap	Time
I	−0.075	1047.32	2.691	1847.936	5.601	6.748	174.054	0.382	0.742	1020.411	2.435	3.776	5.003
II	−0.041	978.06	2.003	1663.736	–	–	–	–	–	–	2.47	3.971	3.224
III	–	–	–	–	14.786	16.279	296.442	5.21	7.148	1686.578	5.104	6.776	25.469

Our computational experiments are carried out on a workstation powered by an Intel Core i7-4790 CPU @ 3.6 GHz with a total RAM of 16 GB running an Ubuntu 16.04 64 bit operating system. The performance of Iridis 4 Computing Cluster and Intel i7-4790 is quite similar, so the computational times for the Exact Algorithms are by itself standardized. Chemla et al. (2013a) on the other hand, ran experiments a workstation powered by an AMD Athlon 64 X2 Dual Core 5600+ processor. To compare the runtime performance of the Tabu Search algorithms proposed in Chemla et al. (2013a) and NLNS + VND, an approximation factor is estimated based on the single thread rating values reported in http://www.cpubenchmark.net/compare.php?cmp[]=86&cmp[]=2226. From the above information, it can be concluded that, a workstation powered by an AMD Athlon 5600+ processor is approximately 2.705325444 times slower than a workstation powered by an Intel i7-4790 CPU. Thus, computational times of the Tabu Search Algorithms (TS1 and TS2), are standardized by dividing their reported computational times by 2.705325444.

Table 7 summarizes our experimental results for 1-PDTSP instances. The results (Gap, Best Gap, Avg Gap and Time) are the average of all the combination of $\mathcal{N}$, Q and α for each set. Corresponding breakdowns for each set is provided in Fig. 13. In Fig. 13, the scale of the *y*-axis for Time (seconds), for all 3 sets are in $\log_{10}$ scale. Detailed experimental results for this case study is available from the authors. In our experiments, NLNS + VND was able to find new solutions for 59 out of 148 instances, for which the optimal solution is not known.

- PEA outperforms NLNS + VND in terms of finding higher quality solutions for instances with $|\mathcal{N}| \leqslant 60$. One of the reason being, PEA allows preemption, whereas NLNS + VND does not.

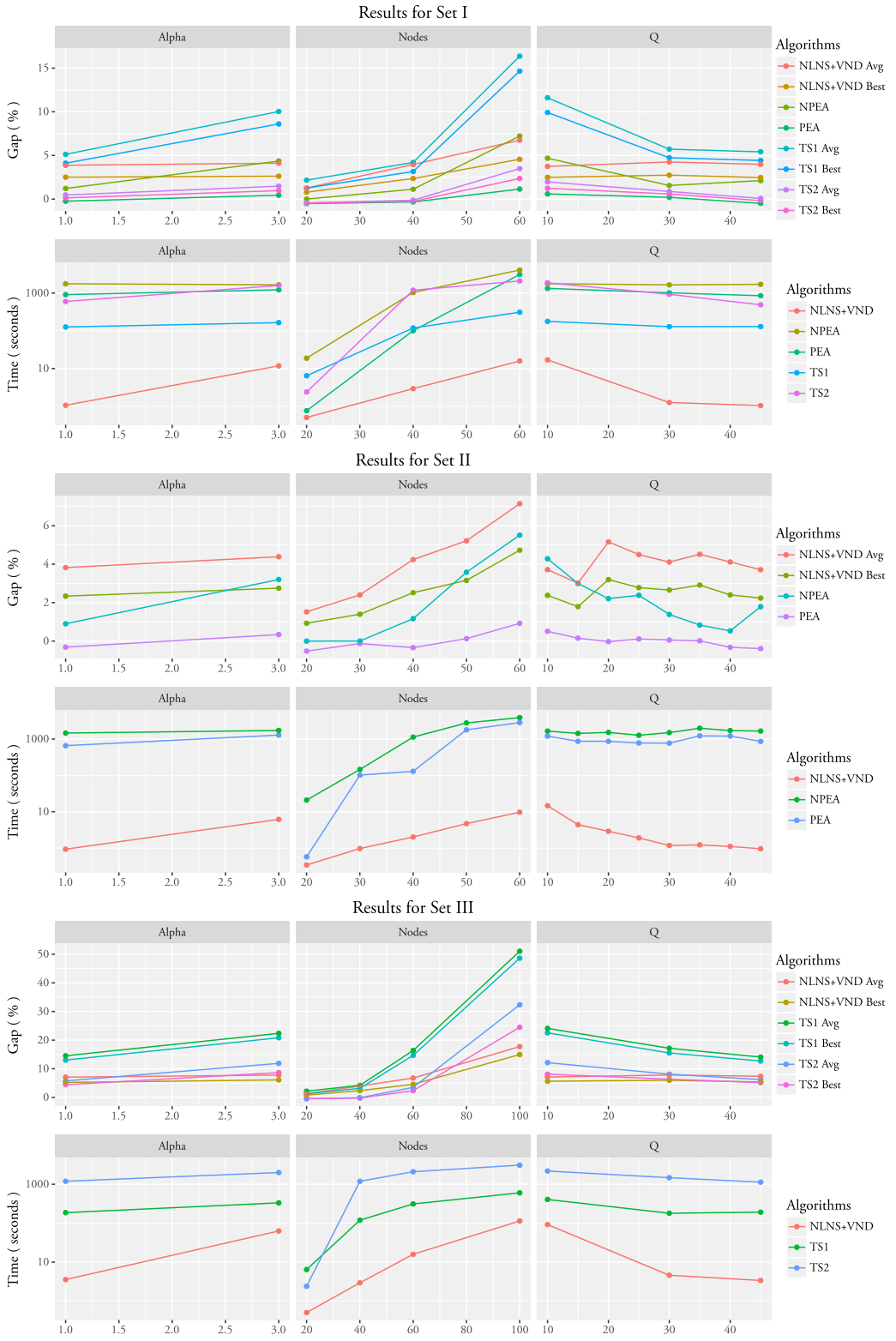

Fig. 13. Summary of results for set I, II and III respectively.

- NLNS + VND is highly competitive with NPEA is terms of quality of solutions found.
- NLNS + VND outperforms TS1 for all instances.
- NLNS + VND is more effective than TS2, for instances with $|\mathcal{N}| > 60$.
- NLNS + VND is the most efficient of all the five algorithms, as on average it has the lowest runtime.
- On average, NLNS + VND is 300, 500, 11 and 66 times faster than PEA, NPEA, TS1 and TS2 respectively.

Other important information not evident from Table 7 but evident from Fig. 13 are:

- NLNS + VND is more effective than the algorithms presented in the literature for realistic instances, i.e., instances with $|\mathcal{N}| \geqslant 50$ and $Q \leqslant 20$.
- NLNS + VND is 3 orders of magnitude faster than both PEA and NPEA, for instances with either $\alpha = 1$ or $Q \geqslant 30$.
- NLNS + VND is 2 orders of magnitude faster than both TS1 and TS2, for instances with $\alpha = 1$ or $Q \geqslant 30$ or $|N| = \{40, 60\}$

7. Case Study 2: Share-A-Bull (SABB) FFBS

The University of South Florida's Tampa campus covers 1700 acres and houses more than 320 buildings. Walking from one building to another during the short breaks between classes is challenging, owing to weather conditions and the heavy weight of textbooks. An annual Tampa campus transportation and parking survey shows that those who drive to campus make, on average, one across-campus trip per day (between buildings or to lunch). Given that there are more than 38,000 students and 1700 faculties and staffs on the Tampa campus, across-campus driving trips can lead to significant fuel consumption and greenhouse gas (GHG) emissions. Thus, USF collaborated with Social Bicycles (SoBi) and developed the Share-A-Bull FFBS program (SABB). Phase I of the program was launched in September 2015 with 100 bicycles. With Phases II and III in next few years, the program will be expanded to 300 bicycles and cover both the Tampa campus and student housing in the vicinity of the campus. The program is expected to be integrated with parking management and other multi-modal transportation initiatives on the campus. SABB provides an excellent case study for our bike rebalancing research.

One of the objectives of this case study is to determine whether SCRP with single and multiple vehicles is feasible for SABB. Another important objective is to determine if NLNS + VND is capable of dealing with increase in complexity of SCRP for FFBS, i.e., when $|\mathcal{N}| \geqslant 100$, $Q \leqslant 10$ and $|\mathcal{V}| > 1$.

Table 8 summarizes the experimental results for five actual rebalancing operations that took place in SABB FFBS. Total Time in Table 8 is the summation of the Computing and the Rebalancing Times. μ and σ in Table 8 are the mean and standard deviation of the respective variable. From Table 8, it is evident that even for a small scale FFBS, SCRP can be computationally challenging. Algorithms proposed in Chemla et al. (2013a), Erdoan et al. (2015) and Alvarez-Valdes et al. (2016) are not capable for handling the corresponding SCRPs except for the first instance with $|\mathcal{N}| = 43$. For each instance, ten trials of NLNS + VND (with Nearest Neighbor 2() as the initial solution creator) are taken, with a fleet of two vehicles each with a capacity $\mathcal{Q}$ of five. The average speed of the rebalancing vehicles was varied between $\{10, 15\}$ miles per hour, and the average loading unloading time per bike ($\overline{\tau}$) was varied between $\{30, 60\}$ seconds. Each of the these rebalancing operations were completed in less than two hours by a fleet of two vehicles each with a capacity $\mathcal{Q}$ of five in real life. This also shows that the accuracy of NLNS + VND in computing the makespan is also very high.

Originally, there were 147 nodes (or ordinary bike racks) with a total capacity to hold 1688 bikes, on the USF Tampa campus. However, an additional 10 nodes (SoBi Hubs) were added at important locations on the USF Tampa campus, to increase user satisfaction. On top of this 157 (original) nodes present in the USF Tampa campus, an additional 293 (artificial) nodes were created at realistic locations on the USF Tampa campus, to simulate parking of bikes outside of SoBi Hubs and ordinary bike racks. The capacities of these artificial nodes were generated from a normal distribution whose mean and standard deviation equals the mean and standard deviation of the capacities of the original 157 nodes. Campus recreation center is the Depot of SABB. Test cases are designed to simulate Phase I, II and III of the Share-A-Bull program, i.e., for 100, 200 and 300 bikes in the network respectively. For each Phase, the maximum $|\mathcal{N}| = \min\{2 \times |\mathcal{B}|, |\mathcal{B}| + 157\}$. Thus for 100, 200 and 300 bikes the maximum $|\mathcal{N}|$ are 200, 357 and 457 nodes respectively.

Table 8
Summary of overall results for SABB real instances.

$\mathcal{B}$	$\mathcal{N}$	Makespan (sec)		Computing time (sec)		Total time (h)	
		μ	ρ	μ	ρ	μ	ρ
42	43	3512	845.13	0.71	0.31	0.98	0.23
44	79	3670.75	845.57	1.37	0.29	1.02	0.23
51	98	4207.5	989.6	2.13	1.05	1.17	0.27
57	96	4358.25	1040.29	4.56	1.99	1.21	0.29
63	118	4842.75	1182.68	6.1	2.62	1.35	0.33

For configurations with $|\mathcal{N}| \leqslant 157$, $|N| - 1$ nodes were selected randomly from the set of 156 original nodes (excluding the Depot). Otherwise all the original 156 nodes (excluding the depot) were selected, and the remaining $|\mathcal{N}| - 157$ nodes, were selected randomly from the 293 artificial nodes. The distance matrix (c_{ij}) was computed from an Open Street Map of the USF, Tampa campus. Loading-unloading operations (d_i) at each node was randomly assigned such that the following conditions are satisfied:

$$d_1 = 0 \tag{29}$$
$$d_i \neq 0, \quad \forall i \in \mathcal{N} \setminus \{1\} \tag{30}$$
$$|d_i| \leqslant \text{Number of Bikes Node i can hold}, \quad \forall i \in \mathcal{N} \setminus \{1\} \tag{31}$$
$$\sum_{i \in \mathcal{N}} d_i = 0 \tag{32}$$
$$\sum_{i \in \mathcal{N}} |d_i| = 2 \times |\mathcal{B}| \tag{33}$$

In total 9 such independent instances were created. These instances are available at https://github.com/aritrasep/BSSLib.jl.

Let us denote the make-span of a fleet of $\mathcal{V}$ rebalancing vehicles be $\mathcal{M}_{|\mathcal{V}|}$. Then we can approximate

$$\mathcal{M}_{|\mathcal{V}|} \approx \frac{\text{Traveling Time if fleet consist of 1 vehicle} + \text{Number of Operations} \times \tau}{|\mathcal{V}|}$$

Thus for a fixed $|\mathcal{V}|$ and τ, $\mathcal{M}_{|\mathcal{V}|}$ will be close to its maximum value, when the **Traveling Time if fleet consist of 1 vehicle** and **Number of Operations** is maximum or pretty close to their maximum values. From Eq. (33), one can see that, the total number of (loading and unloading) operations $= 2 \times |\mathcal{B}|$, which is the maximum possible operation for a fixed $|\mathcal{B}|$, i.e., pickup and drop off of every bike in the system. Further, Eq. (30) ensures that there is at least an unit (positive or negative) imbalance is associated with each node (other than the depot) in the network. This ensures that all nodes in the network is visited at least once, by one or more of the rebalancing vehicles, if not more depending on their respective imbalance. This ensures, that the Traveling Time is also pushed to its maximum value. Thus, the test cases generated simulate extreme scenarios, so that the performance of NLNS + VND and feasibility of SCRP for SABB can be tested in the worst possible scenario.

Given, capacity of the rebalancing vehicle $\mathcal{Q} \in \{5, 10\}$, average speed of the rebalancing vehicle $\in \{10, 15\}$ miles per hour, and average loading unloading time per bike $(\bar{\tau}) \in \{30, 60\}$ seconds, there are a total of 72 different configurations. Ten trials of NLNS + VND (with Nearest Neighbor 2() as the initial solution creator) are taken for each configuration. For each configuration, $|\mathcal{V}| = \frac{|\mathcal{N}|}{100}$. The summary of the experimental results are reported in Table 9.

Total Time in Table 9 is the summation of the Computing and the Rebalancing Times. μ and σ in Table 9 are the mean and standard deviation of the respective major column. From Table 9, it is evident that SCRP is feasible for SABB, as the Total Time is less than 6 h (time available per day for computation and rebalancing) for all the configurations and NLNS + VND is able to compute high quality solutions of SCRP for FFBS in a short period of CPU Time.

Table 9
Summary of overall results for SABB general instances.

$\mathcal{B}$	$\mathcal{N}$	$\mathcal{V}$	Q	Makespan (sec)		Computing time (sec)		Total time (h)	
				μ	σ	μ	σ	μ	σ
100	100	1	5	13519.5	3175.51	32.99	13.24	3.76	0.88
			10	12426.62	3121.92	15.17	6.11	3.46	0.87
	200		5	16769.1	3427.65	40.21	12.58	4.67	0.95
			10	16584.18	3401.45	18.64	9.56	4.61	0.94
200	100	2	5	12774.75	3070.98	22.82	10.4	3.55	0.85
			10	11750.3	3114.54	8.29	3.07	3.27	0.87
	200		5	15426.42	3394.37	27.58	14.42	4.29	0.94
			10	14535.1	3313.93	13.4	7.22	4.04	0.92
	300		5	16526.18	3542.88	65.8	37.97	4.61	0.98
			10	15562	3263.39	28.25	9.54	4.33	0.91
300	100	3	5	12215.9	3112.13	18.39	6.68	3.4	0.86
			10	11346.32	3081.06	5.21	2.04	3.15	0.86
	200		5	14842.05	3286.22	17.26	6.94	4.13	0.91
			10	13685.98	3205.2	7.24	3.42	3.8	0.89
	300		5	14778.64	3306.76	47.64	17.61	4.12	0.92
			10	13933.52	3243.78	19.35	8.64	3.88	0.9
	400		5	15803.7	3365.43	80.76	44.22	4.41	0.94
			10	14832.58	3381.04	38.94	20	4.13	0.94

Table 10
Summary of overall results for Divvy SBBS instances.

$\mathcal{B}$	$\mathcal{N}$	$\mathcal{V}$	Q	Makespan (sec)		Computing time (sec)		Total time (h)	
				μ	σ	μ	σ	μ	σ
500	450	5	10	18009.47	5119.49	37.1	14.2	5.01	1.42
			20	17840.97	4999.4	20.87	8.27	4.96	1.39
1000		10	10	16790.53	5104.62	29.29	4.59	4.67	1.42
			20	16491.13	5070.54	15.18	2.3	4.59	1.41
1500		15	10	15807.07	5096.04	31.55	3.24	4.4	1.42
			20	15970.4	5124.05	17.86	0.75	4.44	1.42
2000		20	10	15768.13	5127.47	40.53	4.1	4.39	1.42
			20	15781.7	5163.12	23.15	0.83	4.39	1.43
2500		25	10	15268.4	5093.73	54.41	6.02	4.26	1.41
			20	15306.67	5131.3	28.2	0.93	4.26	1.43
3000		30	10	15119.53	5069.09	65.62	5.75	4.22	1.41
			20	15136.53	5113.23	34.44	1.23	4.21	1.42

8. Case Study 3: Divvy SBBS

Divvy is a large scale SBBS system in the city of Chicago with 476 nodes (stations with a total capacity to hold approximately 7900 bikes), and 4760 bikes. The objectives for conducting this case study are two fold, first, to determine if SCRP with multiple vehicles is feasible for Divvy and second, to determine if NLNS + VND is capable of dealing with increase in complexity of SCRP for large scale SBBS, i.e., when $|\mathcal{N}| > 400,\ 500 \leqslant |\mathcal{B}|$ and $5 \leqslant |\mathcal{V}|$. The test cases are created using the same method used for creating the SABB general instances as described in Section 7. In total 6 such independent test cases were created. For each test case generated, capacity of the rebalancing vehicle $\mathcal{Q} \in \{10, 20\}$, average speed of the rebalancing vehicle $\in \{40, 50\}$ miles per hour, and average loading unloading time per bike $(\overline{\tau}) \in \{30, 60, 90\}$ seconds, creating a total of 72 different configurations. Five trials of NLNS + VND (with Nearest Neighbor 2() as the initial solution creator) was taken for each configuration. For each configuration, $|\mathcal{V}| = \frac{|\mathcal{N}|}{100}$. These instances are available at https://github.com/aritrasep/BSSLib.jl. The summary of the experimental results are reported in Table 10.

Total Time in Table 10 is the summation of the Computing and the Rebalancing Times. μ and σ in Table 10 are the mean and standard deviation of the respective major column. From Table 10, it is evident that SCRP is also feasible for Divvy instances, as the Total Time is less than 7 h for all configurations and NLNS + VND is able to compute high quality solutions for SCRP for large scale Bike Sharing Systems in a short period of CPU Time.

9. Conclusion

In our study, a Novel MILP for formulating SCRP in FFBS and SBBS based on spacial decomposition is reported. The proposed formulation, can not only handle single and multiple vehicles, but also allows for multiple visits to a node by the same vehicle. The proposed formulation is computationally intractable even for small scale instances owing to the presence of Big M, used for subtour elimination in the constraints. It makes the linear programming relaxation of the formulation extremely weak. Another reason for the computational intractability of the formulation is the significant increase in the number of decision variables owing to spacial decomposition.

A hybrid nested large neighborhood search with variable neighborhood descent algorithm (NLNS + VND) for solving SCRP both effectively and efficiently for FFBS and SBBS is also presented. Computational experiments on 1-PDTSP instances, previously used the literature, demonstrate that NLNS + VND outperforms tabu search and is highly competitive with exact algorithms reported in the literature. The fact that NLNS + VND was able to find all solutions for 148 instances, with 59 of which did not have optimal solutions from applying solution algorithms in existing literature show that, it is more robust than the algorithms previously reported in the literature. Further, NLNS + VND is, on average, 300 times faster than the exact algorithm that allows preemption and 500 times faster than the exact algorithm does not allow preemption. To the best of our knowledge, NLNS + VND is the rst to solve SCRP, with instances having nodes greater than or equal to 50 and vehicle capacity less than 30 both effectively and efficiently. Computational experiments on the new SABB real and general instances (consisting of up to 400 nodes, 300 bikes, and a eet size of up to 3 vehicles) and Divvy instances (consisting of 450 stations, 3000 bikes, and a eet size of up to 30 vehicles), demonstrate that NLNS + VND is able to deal with the increase in scale of SCRP for both FFBS and SBBS. It also shows that SCRP is feasible for both SABB program at USF, Tampa and Divvy SBBS at Chicago.

In future research, we consider strengthening the linear programming relaxation of our proposed formulation by extending valid inequalities proposed in the literature for m-TSP, 1-PDTSP and Q-TSP to our proposed formulation. To deal with increase in the number of variables, strategies based on column generation will be explored. Other interesting strategies can be using the high quality solution provided by NLNS + VND as a starting solution (upper bound) for MIP solvers once some mechanism for strengthening the formulation has been implemented. We also study partial rebalancing in FFBS and SBBS with single and multiple vehicles.

Acknowledgements

This study was supported by the Student Green Energy Fund (SGEF) at University of South Florida. The authors thank Joseph Fields for creating an Open Street Map of the USF Tampa campus and for designing the User Interface of the USF Rebalancing App. We also greatly appreciate the three anonymous reviewers, for providing in-depth comments, suggestions, and corrections on an earlier draft of this paper.

References

Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P., 2002. A survey of very large-scale neighborhood search techniques. Discr. Appl. Math. 123, 75–102. http://dx.doi.org/10.1016/S0166-218X(01)00338-9 <http://www.sciencedirect.com/science/article/pii/S0166218X01003389>.

Alvarez-Valdes, R., Belenguer, J.M., Benavent, E., Bermudez, J.D., Muoz, F., Vercher, E., Verdejo, F., 2016. Optimizing the level of service quality of a bike-sharing system. Omega 62, 163–175. http://dx.doi.org/10.1016/j.omega.2015.09.007 <http://www.sciencedirect.com/science/article/pii/S0305048315002005>.

Anily, S., Bramel, J., 1999. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. Naval Res. Logist. (NRL) 46, 654–670. http://dx.doi.org/10.1002/(SICI)1520-6750(199909)46:6<654::AID-NAV4>3.0.CO;2-A.

Applegate, D., Cook, W., Rohe, A., 2003. Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. Comput. 15, 82–92. http://dx.doi.org/10.1287/ijoc.15.1.82.15157.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34, 209–219. http://dx.doi.org/10.1016/j.omega.2004.10.004 <http://www.sciencedirect.com/science/article/pii/S0305048304001550>.

Bezanson, J., Karpinski, S., Shah, V.B., Edelman, A., 2012. Julia: A Fast Dynamic Language for Technical Computing. Available from: <arXiv preprint arXiv: 1209.5145>.

Boyacı, B., Zografos, K.G., Geroliminis, N., 2015. An optimization framework for the development of efficient one-way car-sharing systems. Eur. J. Oper. Res. 240, 718–733.

Chalasani, P., Motwani, R., 1999. Approximating capacitated routing and delivery problems. SIAM J. Comput. 28, 2133–2149. http://dx.doi.org/10.1137/S0097539795295468.

Chemla, D., Meunier, F., Calvo, R.W., 2013a. Bike sharing systems: solving the static rebalancing problem. Discr. Optim. 10, 120–146. http://dx.doi.org/10.1016/j.disopt.2012.11.005 <http://www.sciencedirect.com/science/article/pii/S1572528612000771>.

Chemla, D., Meunier, F., Pradeau, T., Calvo, R.W., 2013b. Self-service Bike Sharing Systems: Simulation, Repositioning, Pricing.

Contardo, C., Morency, C., Rousseau, L.M., 2012. Balancing a Dynamic Public Bike-sharing System, vol. 4. CIRRELT.

Dell'Amico, M., Hadjicostantinou, E., Iori, M., Novellani, S., 2014. The bike sharing rebalancing problem: mathematical formulations and benchmark instances. Omega 45, 7–19. http://dx.doi.org/10.1016/j.omega.2013.12.001 <http://www.sciencedirect.com/science/article/pii/S0305048313001187>.

DellAmico, M., Iori, M., Novellani, S., Sttzle, T., 2016. A destroy and repair algorithm for the bike sharing rebalancing problem. Comput. Oper. Res. 71, 149–162. http://dx.doi.org/10.1016/j.cor.2016.01.011 <http://www.sciencedirect.com/science/article/pii/S0305054816300016>.

DeMaio, P., 2009. Bike-sharing: history, impacts, models of provision, and future. J. Public Transp. 12, 3.

Erdoan, G., Battarra, M., Calvo, R.W., 2015. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. Eur. J. Oper. Res. 245, 667–679. http://dx.doi.org/10.1016/j.ejor.2015.03.043 <http://www.sciencedirect.com/science/article/pii/S0377221715002659>.

Erdoan, G., Laporte, G., Calvo, R.W., 2014. The static bicycle relocation problem with demand intervals. Eur. J. Oper. Res. 238, 451–457. http://dx.doi.org/10.1016/j.ejor.2014.04.013 <http://www.sciencedirect.com/science/article/pii/S0377221714003221>.

Forma, I.A., 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. http://dx.doi.org/10.1016/j.trb.2014.10.003 <http://www.sciencedirect.com/science/article/pii/S0191261514001726>.

Ghilas, V., Demir, E., Woensel, T.V., 2016. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. Comput. Oper. Res. 72, 12–30. http://dx.doi.org/10.1016/j.cor.2016.01.018 <http://www.sciencedirect.com/science/article/pii/S0305054816300144>.

Helsgaun, K., 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. 126, 106–130. http://dx.doi.org/10.1016/S0377-2217(99)00284-2 <http://www.sciencedirect.com/science/article/pii/S0377221799002842>.

Helsgaun, K., 2009. General k-opt submoves for the Lin–Kernighan TSP heuristic. Math. Programm. Comput. 1, 119–163. http://dx.doi.org/10.1007/s12532-009-0004-6.

Hernndez-Prez, H., Salazar-Gonzlez, J.J., 2004. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discr. Appl. Math. 145, 126–139. http://dx.doi.org/10.1016/j.dam.2003.09.013 (graph Optimization {IV}) <http://www.sciencedirect.com/science/article/pii/S0166218X0400071X>.

Ho, S.C., Szeto, W., 2014. Solving a static repositioning problem in bike-sharing systems using iterated tabu search. Transp. Res. Part E: Logist. Transp. Rev. 69, 180–198. http://dx.doi.org/10.1016/j.tre.2014.05.017 <http://www.sciencedirect.com/science/article/pii/S1366554514000945>.

Ho, S.C., Szeto, W., 2017. A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. Transp. Res. Part B: Methodol. 95, 340–363. http://dx.doi.org/10.1016/j.trb.2016.11.003 <http://www.sciencedirect.com/science/article/pii/S0191261516302065>.

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: 14th European Conference, EvoCOP 2014, Granada, Spain, April 23–25, 2014, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 73–84. http://dx.doi.org/10.1007/978-3-662-44320-0_7.

Laporte, G., Meunier, F., Wolfler Calvo, R., 2015. Shared mobility systems. 4OR 13, 341–360. http://dx.doi.org/10.1007/s10288-015-0301-z.

Laporte, G., Nobert, Y., 1980. A cutting planes algorithm for the m-salesmen problem. J. Oper. Res. Soc. 31, 1017–1023. http://dx.doi.org/10.1057/jors.1980.188.

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, 1567–1578. http://dx.doi.org/10.1109/TITS.2014.2303986.

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, 597–629. http://dx.doi.org/10.1007/s10898-014-0147-5.

Raviv, T., Tzur, M., Forma, I.A., 2013. Static repositioning in a bike-sharing system: models and solution approaches. EURO J. Transp. Logist. 2, 187–229. http://dx.doi.org/10.1007/s13676-012-0017-6.

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. http://dx.doi.org/10.1016/j.tre.2014.10.005 <http://www.sciencedirect.com/science/article/pii/S1366554514001720>.

Reiss, S., Bogenberger, K., 2015. GPS-data analysis of munich's free-floating bike sharing system and application of an operator-based relocation strategy. In: Proceedings of the 2015 IEEE 18th International Conference on Intelligent Transportation Systems. IEEE Computer Society, Washington, DC, USA, pp. 584–589. http://dx.doi.org/10.1109/ITSC.2015.102.

Schuijbroek, J., Hampshire, R., van Hoeve, W.J., 2017. Inventory rebalancing and vehicle routing in bike sharing systems. Eur. J. Oper. Res. 257, 992–1004. http://dx.doi.org/10.1016/j.ejor.2016.08.029 <http://www.sciencedirect.com/science/article/pii/S0377221716306658>.

Singla, A., Santoni, M., Bartók, G., Mukerji, P., Meenen, M., Krause, A., 2015. Incentivizing users for balancing bike sharing systems. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. AAAI Press, pp. 723–729 <http://dl.acm.org/citation.cfm?id=2887007.2887108>.

Szeto, W., Liu, Y., Ho, S.C., 2016. Chemical reaction optimization for solving a static bike repositioning problem. Transp. Res. Part D: Transp. Environ. 47, 104–135. http://dx.doi.org/10.1016/j.trd.2016.05.005 <http://www.sciencedirect.com/science/article/pii/S1361920916300608>.
Toth, P., Vigo, D., 2003. The granular tabu search and its application to the vehicle-routing problem. INFORMS J. Comput. 15, 333–346. http://dx.doi.org/10.1287/ijoc.15.4.333.24890.
Weikl, S., Bogenberger, K., 2013. Relocation strategies and algorithms for free-floating car sharing systems. IEEE Intell. Transp. Syst. Mag. 5, 100–111.