Balancing Bicycle Sharing Systems: A Variable Neighborhood Search Approach*

Marian Rainer-Harbach, Petrina Papazek, Bin Hu, and Günther R. Raidl**

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{rainer-harbach|papazek|hu|raidl}@ads.tuwien.ac.at

Abstract. We consider the necessary redistribution of bicycles in public bicycle sharing systems in order to avoid rental stations to run empty or entirely full. For this purpose we propose a general Variable Neighborhood Search (VNS) with an embedded Variable Neighborhood Descent (VND) that exploits a series of neighborhood structures. While this metaheuristic generates candidate routes for vehicles to visit unbalanced rental stations, the numbers of bikes to be loaded or unloaded at each stop are efficiently derived by one of three alternative methods based on a greedy heuristic, a maximum flow calculation, and linear programming, respectively. Tests are performed on instances derived from real-world data and indicate that the VNS based on a greedy heuristic represents the best compromise for practice. In general the VNS yields good solutions and scales much better to larger instances than two mixed integer programming approaches.

1 Introduction

A large number of public Bicycle Sharing Systems (BSSs) has been introduced in many cities around the world in the last decade. Such systems augment public transport well and frequently present attractive "green" alternatives to individual motorized traffic. A BSS consists of a number of stations where users can rent and return bikes in an automated way. Operators face an important challenge with regard to customer satisfaction: Due to different factors such as the topographical height, the numbers of bikes rented and returned, respectively, differ significantly among the stations. Running such a system without any maintenance would therefore soon result in many completely empty or, equally worse, completely full stations. Thus the operator needs to actively rebalance the system by moving bicycles between stations with a fleet of vehicles, e.g. cars with trailers. In the *Balancing Bicycle Sharing System* (BBSS) problem we aim at finding efficient vehicle routes with corresponding bicycle-loading instructions at the visited stations in order to bring the system in balance as far as possible.

In this work we address this problem by a Variable Neighborhood Search (VNS) with an embedded Variable Neighborhood Descent (VND), which exploit various specifically

* This work is supported by the Austrian Research Promotion Agency (FFG), contract 831740.
** The authors thank Matthias Prandtstetter, Andrea Rendl and Markus Straub from the Austrian Institute of Technology (AIT) for the collaboration in this project.

designed neighborhood structures. While tours are searched within the VNS/VND, corresponding loading instructions are efficiently derived by either a greedy approach, a maximum flow calculation, or linear programming. Experiments are performed on benchmark instances derived from Citybike, the BSS in Vienna, Austria. They indicate that high quality solutions can be found with this approach and that the maximum flow based calculating of loading instruction performs best in practice.

2 The Balancing Bicycle Sharing System Problem

In this section we formalize the BBSS problem. Currently, we only consider a static problem variant in which user activities during the rebalancing process are neglected. The BSS is represented by a complete directed graph $G_0 = (V_0, A_0)$. Node set $V_0 = V \cup O$ consists of nodes for the rental stations V as well as start and end points O for vehicles (garages or overnight parking places). Each arc $(u,v) \in A_0$ has associated a travel time $t_{u,v} > 0$ that includes an expected time for parking near v and loading/unloading bikes. Let the subgraph induced by the bike stations V only be $G = (V,A)$, $A \subset A_0$.

Each station $v \in V$ has associated a capacity $C_v \geq 0$, i.e., the number of available parking positions, the number of available bikes at the beginning of the rebalancing process $p_v \geq 0$, and a target number of available bikes after rebalancing $q_v \geq 0$. A fleet of vehicles $L = \{1,\ldots,|L|\}$ is available for transporting bikes. Each vehicle $l \in L$ has a capacity of $Z_l > 0$ bikes, a total time budget $\hat{t}_l$ within which it has to finish a route (i.e., the worker's shift length), as well as specific start and destination nodes $s_l, d_l \in O$, respectively. We assume that all vehicles start and finish rebalancing empty. A solution consists of two parts. The first part is the route for each vehicle $l \in L$ specified by an ordered sequence of visited stations $r_l = (r_l^1,\ldots,r_l^{\rho_l})$ with $r_l^i \in V$, $i = 1,\ldots,\rho_l$ and ρ_l representing the number of stops. Note that stations may be visited multiple times by the same or different vehicles. Start and end points s_l and d_l are fixed for each vehicle and are prepended and appended, respectively, to each route in order to obtain complete tours. The second part consists of loading and unloading instructions $y_{l,v}^{+,i}, y_{l,v}^{-,i} \in \{0,\ldots,Z_l\}$ with $l \in L$, $v \in V$, and $i = 1,\ldots,\rho_l$, specifying how many bikes are picked up or delivered, respectively, at station v at the i-th stop of vehicle l. Note that $\forall v \neq r_l^i : \; y_{l,v}^{+,i} = y_{l,v}^{-,i} = 0$.

The following conditions must hold in a feasible solution: The number of bikes available at each station $v \in V$ needs to be within $\{0,\ldots,C_v\}$. For any vehicle $l \in L$ capacities Z_l may never be exceeded, and the tour time t_l

$$t_l = \begin{cases} t_{s_l,r_l^1} + \sum_{i=2}^{\rho_l} t_{r_l^{i-1},r_l^i} + t_{r_l^{\rho_l},d_l} & \text{for } \rho_l > 0 \\ t_{s_l,d_l} & \text{for } \rho_l = 0, \end{cases} \tag{1}$$

is restricted by the time budget $\hat{t}_l$, $\forall l \in L$. Let a_v be the final number of bikes after rebalancing at each station $v \in V$,

$$a_v = p_v + \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y_{l,v}^{-,i} - y_{l,v}^{+,i} \right). \tag{2}$$

The objective is to find a feasible solution that primarily minimizes the deviation from the target number of bikes $\delta_v = |a_v - q_v|$ at each station $v \in V$ and secondarily the

number of loading/unloading activities plus the overall time required for all routes, i.e.,

$$\min \quad \alpha^{\text{bal}} \sum_{v \in V} \delta_v + \alpha^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y^{+,i}_{l,r^i_l} + y^{-,i}_{l,r^i_l} \right) + \alpha^{\text{work}} \sum_{l \in L} t_l, \tag{3}$$

where $\alpha^{\text{bal}}, \alpha^{\text{load}}, \alpha^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms.

A simplification that may be exploited in different approaches is to consider *monotonicity* regarding fill levels of stations. Let $V_{\text{pic}} = \{v \in V \mid p_v \geq q_v\}$ denote pickup stations and $V_{\text{del}} = \{v \in V \mid p_v < q_v\}$ denote delivery stations. A vehicle is only allowed to load bicycles at pickup stations and unload them at delivery stations. In this way the number of bikes decreases or increases monotonically, therefore the order in which different vehicles visit a single station does not matter. On the downside, forcing monotonicity might exclude some better solutions that would have been feasible without this restriction.

3 Related Work

It was not until recently that the BBSS problem has been recognized as a combinatorial optimization problem and the operations research community described a few systematic solution approaches. However, they address significantly different problem variants and a direct comparison between existing approaches is difficult. The majority of existing works uses mixed integer programming (MIP), which in principle is able to find proven optimal solutions but in practice is restricted to very small instances.

Chemla et al. [1] address the static case with only one vehicle and achieving perfect balance as a hard constraint. They describe a branch-and-cut approach utilizing an embedded tabu search for locally improving incumbent solutions. To the best of our knowledge, their tabu search is the only metaheuristic approach applied to the rebalancing problem until now. One of the key concepts is to only consider the visiting order of the rebalancing vehicle in the solution representation and to obtain the loading instructions by an auxiliary algorithm based on a maximum flow computation. With this technique the search space can be reduced significantly. In our work we extend this idea towards our more general problem definition. Raviv et al. [2] propose four MIP models. In their objective function they model user dissatisfaction and tour lengths but ignore the number of loading operations. The models were tested on real-world data obtained from Vélib (Paris) with up to 60 stations. Results show that the most basic arc indexed model produces the best lower bounds in a given time limit, but more complex models offer more flexibility with respect to the requirements. Benchimol et al. [3] again assume balancing as hard constraint, only consider the total tour length as objective, and focus on approximation algorithms for selected special situations. Finally, Contardo et al. [4] investigate the more complex dynamic scenario where rebalancing is done while the bike sharing system is in use. They propose an arc-flow formulation and a pattern-based formulation for a space-time network model. The latter is solved heuristically by a hybrid approach using column generation and Benders decomposition. On randomly created instances, this approach was able to handle instances with up to 100 stations and 60 time periods, however significant gaps between lower and upper bounds still remain.

There are other works in the literature which focus on the strategic planning aspects of bike sharing systems (i.e., location and network design depending on demands). However, these aspects are not within the scope of this work. More generally, BBSS is closely related to diverse variants of the classical vehicle routing problem (VRP). However, it differs in substantial ways: Most importantly, stations may be visited multiple times, even by different vehicles. Consequently, BBSS can be described as a capacitated single commodity split pickup and delivery VRP.

4 Greedy Construction Heuristic

To efficiently generate a meaningful initial solution, we employ a construction heuristic based on greedy principles. This procedure assumes monotonicity as described in Section 2. A solution is built by iteratively creating a tour for each vehicle following a local best successor strategy. From the last station of a partial tour, we first determine the *set $F \subseteq V$ of feasible successor stations*. These are all stations that are not yet balanced and can be reached without exceeding the shift length, i.e., there is enough time left to visit the station and to go to the destination node afterwards.

For each such candidate station $v \in F$, we calculate the maximum amount of bicycles that can be picked up or delivered by

$$\gamma_v = \begin{cases} \min(a_v - q_v, Z_l - b_l) & \text{for } v \in F \cap V_{\text{pic}}, \\ \min(q_v - a_v, b_l) & \text{for } v \in F \cap V_{\text{del}}, \end{cases} \tag{4}$$

where b_l represents the final load of vehicle l and a_v the final number of bikes at station v in the currently considered partial tour. For $\rho_l = 0$ they are initialized with $b_l = 0$ and $a_v = p_v$.

We assume that no bikes are allowed to remain on a vehicle when returning to the depot. Therefore, an additional correction is necessary for pickup stations: We determine for each $v \in F \cap V_{\text{pic}}$ if after visiting v the remaining time budget allows the vehicle to deliver at least $b_l + 1$ bicycles to other stations, i.e., all bikes the vehicle currently has loaded plus at least one that would be picked up from v. If this is not the case, visiting v is useless as no bike may finally be picked up there.

For this purpose, we estimate the number of deliverable bikes b_v^{del} after visiting v by iteratively applying the exact same greedy heuristic restricted to delivery stations only. We stop extending this delivery-only route when either $b_v^{\text{del}} \geq b_l + \gamma_v$ (i.e., we have shown that all bicycles picked up at v can be delivered later) or the time budget $\hat{t}_l$ is exceeded. Then, pickup stations v with $b_v^{\text{del}} < b_l + 1$ are removed from set F, while the number of bikes to be picked up is possibly reduced for the others:

$$\gamma_v \leftarrow \min(\gamma_v, b_v^{\text{del}} - b_l), \quad \forall v \in F \cap V_{\text{pic}}. \tag{5}$$

Now, all candidate stations $v \in F$ are evaluated using the ratio $\gamma_v / t_{u,v}$, where $t_{u,v}$ is the traveling time from the vehicle's last location u to station v; thus we consider the balance gain per time unit. The node $v \in F$ with the highest ratio is then appended to the

tour of vehicle l; ties are broken randomly. Loading instructions are set as follows:

$$y_{l,v}^{+,\rho_l} = \gamma_v \text{ and } y_{l,v}^{-,\rho_l} = 0 \qquad \text{if } v \in V_{\text{pic}}, \tag{6}$$

$$y_{l,v}^{+,\rho_l} = 0 \text{ and } y_{l,v}^{-,\rho_l} = \gamma_v \qquad \text{if } v \in V_{\text{del}}. \tag{7}$$

Next, b_l and a_v are updated accordingly and the procedure continues with the next extension, evaluating stations in F from scratch, until no feasible extension remains.

5 Variable Neighborhood Search

In this section we describe our VNS approach. It uses the general VNS scheme with an embedded VND for local improvement as described in [5].

5.1 Solution Representation and Derivation of Loading Instructions

Concerning the VNS we use an incomplete solution representation by storing for each vehicle $l \in L$ its route $r_l = (r_l^1, \ldots, r_l^{\rho_l})$ only. Corresponding loading instructions $y_{l,v}^{+,i},\ y_{l,v}^{-,i},\ l \in L,\ v \in V,\ i = 1, \ldots, \rho_l$ are derived for each created set of tours by one of the following, alternative procedures, which have different assets and drawbacks.

Greedy Heuristic (GH): This simplest and fastest approach follows the pure greedy strategy from the construction heuristic and assumes monotonicity. For each tour, the stations are considered in the order as they are visited and loading instructions are computed as described in Section 4. Even under the restriction of monotonicity, GH is not guaranteed to find optimal loading instructions. For example, it can be beneficial to retain bikes in the vehicle at a first stop at some station v in order to satisfy a following delivery station as v will be visited a second time and can also be satisfied then.

Maximum Flow Approach (MF): When assuming monotonicity, we are able to derive optimal loading instructions via an efficient maximum flow computation on a specifically defined flow network. This approach is inspired by [1] but extends their method towards multiple vehicles and the consideration of balance in the objective function. We define graph $G_{\text{fm}} = (V_{\text{fm}}, A_{\text{fm}})$ with node set $V_{\text{fm}} = \{\sigma, \tau\} \cup V_{\text{pic}} \cup V_{\text{del}} \cup V_L$ where σ and τ are the source and target nodes of the flow, respectively, and $V_L = \bigcup_{l \in L} V_l$ with $V_l = \{v_l^i \mid l \in L,\ i = 1 \ldots, \rho_l\}$ represents the stops of all routes.

Arc set $A_{\text{fm}} = A_\sigma \cup A_L \cup A_{\text{pic}} \cup A_{\text{del}} \cup A_\tau$ consists of:

- $A_\sigma = \{(\sigma, v) \mid v \in V_{\text{pic}}\}$ with capacities $p_v - q_v$.
- $A_\tau = \{(v, \tau) \mid v \in V_{\text{del}}\}$ with capacities $q_v - p_v$.
- $A_{\text{pic}} = \{(v, v_l^i) \mid v_l^i \in V_L,\ v = r_l^i,\ v \in V_{\text{pic}}\}$, i.e., each pickup node in V_{pic} is connected with every node representing a stop at this station in any route $l \in L$. These arcs' capacities are not limited.
- $A_{\text{del}} = \{(v_l^i, v) \mid v_l^i \in V_L,\ v = r_l^i,\ v \in V_{\text{del}}\}$, i.e., each node representing a stop at a delivery station is connected to the corresponding delivery node in V_{del}. These arcs' capacities are also not limited.
- $A_L = \{(v_l^{i-1}, v_l^i) \mid v_l^i \in V_L,\ i > 1\}$, i.e., the nodes representing the stops in each tour are connected according to the tour. Arc capacities are Z_l.

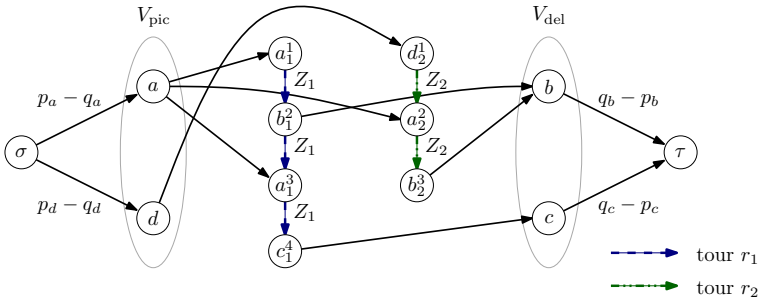

Fig. 1. Exemplary flow network when considering monotonicity for the tours $r_1 = (a,b,a,c)$ and $r_2 = (d,a,b)$ with $V_{\text{pic}} = \{a,d\}$ and $V_{\text{del}} = \{b,c\}$.

An exemplary network is shown in Figure 1. Calculating a maximum (σ,τ)-flow on it directly yields optimal loading instructions $y_{l,v}^{+,i}$, $y_{l,v}^{-,i}$ via the flows on the corresponding arcs A_{pic} and A_{del}, respectively. In our implementation, we used the efficient push-relabel method from Cherkassky and Goldberg [6] for the flow computation.

Linear Programming Approach (LP): Finally, we are able to determine optimal loading instructions even for the general, not necessarily monotonic case by solving a minimum cost flow problem on a differently defined network by linear programming. The main difference is that the order in which vehicles make their stops (at possibly the same stations) is considered. Bikes can be buffered at stations or even be directly transferred from one vehicle to another when they meet.

Let $t(r_l^i)$ denote the absolute time when vehicle l makes its i-th stop at station r_l^i. We define the multi-graph $G_{\text{f}} = (V_{\text{f}}, A_{\text{f}})$ with node set $V_{\text{f}} = \{\sigma,\tau\} \cup V_t$ where $V_t = \{v^j \mid \exists v_l^i \in V_l : t(r_l^i) = j\}$, i.e., besides source and target nodes σ and τ we have a node v^j for each station v and time j when a vehicle arrives at v. Furthermore $V^{\text{first}} = \{v^{j_{\min}} \in V_t \mid j_{\min} = \min\{j \mid v^j \in V_t\}\}$, i.e., these nodes represent the first visits of all stations among all routes, and $V^{\text{last}} = \{v^{j_{\max}} \in V_t \mid j_{\max} = \max\{j \mid v^j \in V_t\}\}$, i.e., these nodes represent the last visits of all stations.

Arc set $A_{\text{f}} = A_\sigma \cup A_\tau \cup A_R \cup A_V$ consists of:

- $A_\sigma = \{(\sigma, v^j) \mid v^j \in V^{\text{first}}\}$ with capacities p_v.
- $A_\tau = \{(v^j, \tau) \mid v^j \in V^{\text{last}}\}$ with capacities q_v.
- $A_R = \bigcup_{l \in L} A_{R,l}$ with $A_{R,l} = \{(u^{j'}, v^j) \mid u = r_l^{i-1},\ v = r_l^i,\ j' = t(r_l^{i-1}),\ j = t(r_l^i),\ i = 2,\ldots,\rho_l\}$, $\forall l \in L$, i.e., the arcs representing the flow induced by the vehicles. Capacities are Z_l. Note that multiple arcs exist between two nodes if two (or more) vehicles leave and arrive at the same stations exactly at the same times.
- $A_V = \bigcup_{v \in V} A_v$ with $A_v = \{(v^{j_1}, v^{j_2}), \ldots, (v^{j_{\max}-1}, v^{j_{\max}})\}$, $(v^{j_1}, \ldots, v^{j_{\max}})$ is the sequence of nodes $\{v^j \in V_t\}$ sorted according to increasing j. Capacities are C_v.

An example of this network is given in Figure 2. Now, a simple maximum flow calculation would in general not yield optimal or even feasible loading instructions. Instead, we have to solve a minimum cost flow problem via the following LP, which uses flow variables $f_{u,v}$, $\forall (u,v) \in A_{\text{f}}$. By $pred_l(v^j) \in V_t$ we denote the predecessor of

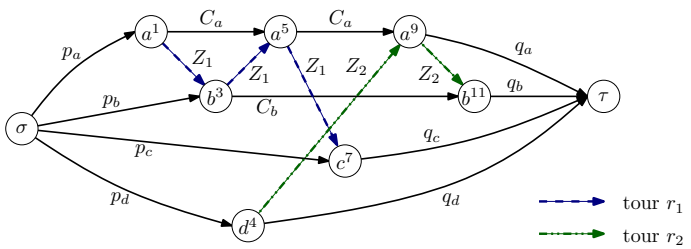

Fig. 2. Exemplary flow network for the general case with tours $r_1 = (a,b,a,c)$ and $r_2 = (d,a,b)$.

the node v^j on the route of vehicle l, i.e., $pred_l(v^j) = u^{j'}$ with $u = v_l^{i-1},\ j' = t(r_l^{i-1})$, and by $succ_l(v^j) \in V_t$ the successor, i.e., $succ_l(v^j) = w^{j''}$ with $w = v_l^{i+1},\ j'' = t(r_l^{i+1})$.

$$\min \quad \alpha^{\text{bal}} \sum_{\forall v \in V^{\text{last}}} \delta_v + \alpha^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left(y_{l,r_l^i}^{+,i} + y_{l,r_l^i}^{-,i} \right) \tag{8}$$

subject to

$$\sum_{(u,v^j) \in A_\sigma \cup A_V} f_{u,v^j} + \sum_{l \in L} \sum_{(u,v^j) \in A_{R,l}} f_{u,v^j} = \sum_{(v^j,w) \in A_\tau \cup A_V} f_{v^j,w} + \sum_{l \in L} \sum_{(v^j,w) \in A_{R,l}} f_{v^j,w} \qquad \forall v^j \in V_t \tag{9}$$

$$y_{l,v}^{+,i} - y_{l,v}^{-,i} = \begin{cases} f_{v^j,succ_l(v^j)} & \forall l \in L,\ i = 1,\ v = r_l^i,\ j = t(r_l^i) \\ f_{v^j,succ_l(v^j)} - f_{pred_l(v^j),v^j} & \forall l \in L, i = 2,\ldots,\rho_l - 1, v = r_l^i, j = t(r_l^i) \\ -f_{pred_l(v^j),v^j} & \forall l \in L,\ i = \rho_l,\ v = r_l^i,\ j = t(r_l^i) \end{cases} \tag{10}$$

$$f_{\sigma,v^j} = p_v \qquad \forall (\sigma, v^j) \in A_\sigma \tag{11}$$

$$f_{v^j,\tau} - q_v \le \delta_v \qquad \forall (v^j,\tau) \in A_\tau \tag{12}$$

$$q_v - f_{v^j,\tau} \le \delta_v \qquad \forall (v^j,\tau) \in A_\tau \tag{13}$$

$$0 \le f_{v^j,\tau} \le C_v \qquad \forall (v^j,\tau) \in A_\tau \tag{14}$$

$$0 \le f_{u^{j'},v^j} \le Z_l \qquad \forall l \in L,\ (u^{j'},v^j) \in A_{R,l} \tag{15}$$

$$0 \le f_{v^{j'},v^j} \le C_v \qquad \forall (v^{j'},v^j) \in A_V \tag{16}$$

$$\delta_v \ge 0 \qquad \forall (v^j,\tau) \in A_\tau \tag{17}$$

$$y_{l,v}^{+,i} \in \{0,\ldots,Z_l\} \qquad \forall l \in L,\ v \in V,\ i = 1,\ldots,\rho_l \tag{18}$$

$$y_{l,v}^{-,i} \in \{0,\ldots,Z_l\} \qquad \forall l \in L,\ v \in V,\ i = 1,\ldots,\rho_l \tag{19}$$

The objective function (8) is directly derived from our main objective (3). Equations (9) are the flow conservation equalities, while equations (10) link the loading instruction variables with the flows. The flows at arcs $(\sigma, v^j) \in A_\sigma$ are fixed to the station's initial number of bikes p_v in (11).

As we have a capacitated but unrestricted flow network with all capacities being integer, the LP is totally unimodular and the corresponding polytope's extreme points are all integer. Therefore by solving this LP with a common LP solver (or more specifically a network simplex algorithm), we obtain optimal integral values for the loading instructions.

5.2 VND and VNS Neighborhood Structures

We use several classical neighborhood structures that were successfully applied in various VRPs together with new structures exploiting specifics of BBSS. Concerning the classical neighborhood structures, we based our design on the experience from [7].

VND Neighborhoods: The following neighborhoods are all searched in a best improvement fashion and applied in the given, static order. Preliminary experiments with a dynamic reordering strategy brought no significant advantages. All created candidate tours are incrementally checked for feasibility with respect to time budgets and infeasible solutions are discarded. For a feasible solution we derive loading instructions by one of the methods from Section 5.1 and remove obsolete nodes without any loading actions.

Remove station (REM-VND): This neighborhood considers all single station removals to avoid unnecessary visits.

Insert unbalanced station (INS-U): This neighborhood includes all feasible solutions where a yet unbalanced station is inserted at any possible position.

Intra-route 2-opt (2-OPT): This is the classical 2-opt neighborhood for the traveling salesman problem, applied individually to each route.

Replace station (REPL): Here, any solution in which one station is replaced by a different, yet unbalanced station is included.

Intra or-opt (OR-OPT): This neighborhood considers all solutions in which sequences of one, two, or three consecutive stations are moved to another place within the same route.

2-opt* inter-route exchange (2-OPT*): This classical neighborhood considers all feasible exchanges of arbitrarily long end segments of two routes.

Intra-route 3-opt (3-OPT): This neighborhood resembles a restricted form of the well-known 3-opt neighborhood, individually applied to each route. For any partitioning of a route into three nonempty subsequences r_l =(a,b,c), the routes (b,a,c) and (a,c,b) are considered. An effective enumeration scheme excludes all solutions of the previous neighborhoods.

VNS Neighborhoods: For diversification, the shaking procedure selects solutions randomly from the following types of VNS neighborhoods, which are all parameterized by δ, yielding a total of 24 individual neighborhoods. During this process, created routes that violate the time budget are repaired by removing stations from the end.

Move sequence (MV-SEQ): Select a sequence of one to $\min(\delta, \rho_l)$ stations at random, delete it, and reinsert it at a random position of a different route. If the original route contains less than δ stations, the whole route is inserted at the target route. Both source and target routes are selected randomly. $\delta \in \{1, \ldots, 5, \rho_l\}$.

Exchange sequence (EX-SEQ): Exchange two randomly selected segments of length one to $\min(\delta, \rho_l)$ between two randomly chosen routes. $\delta \in \{1, \ldots, 5, \rho_l\}$.

Remove stations (REM-VNS): Consider all stations of all routes and remove each station with probability $\delta \in \{10\%, 14\%, 18\%, 22\%, 26\%, 30\%\}$.

Destroy and recreate (D&R): Select a random position in a randomly chosen route, remove all nodes from this position to the end, and recreate a new end segment by applying a randomized version of the greedy construction heuristic. The randomization is done in the typical GRASP-like way [8] with the threshold parameter set to $\delta \in \{0\%, 4\%, 8\%, 12\%, 16\%, 20\%\}$.

6 Computational Results

We tested our VNS algorithm on a set of instances based on real-world data provided by Citybike Vienna[1] which runs a bike-sharing system with 92 stations. They are generated as follows:

- Travel times $t_{u,v},\ (u,v) \in A_0$ are real average driving times plus an estimation for parking the vehicle and loading/unloading bikes based on the experience of the drivers.
- The number of currently available bikes p_v at station $v \in V$ is taken from a snapshot of the system.
- The target value q_v is assumed to be 50% of the station's capacity.
- In order to make *perfect balance* at least theoretically possible when having enough time, $\sum_{v \in V} p_v = \sum_{v \in V} q_v$ must hold. This is established by applying small changes to p_v for some randomly chosen stations.
- We derived instances with $|V| \in \{10, 20, 30, 60, 90\}$ stations by choosing them randomly from the pool of 92 stations. In addition we consider one common depot (one of the remaining stations) to be the start and end point for all vehicles.
- We assume a homogeneous fleet of $|L| \in \{1, 2, 3, 5\}$ vehicles with capacity $Z_l = 20,\ \forall l \in L$.
- The total time budget for each vehicle is set to $\hat{t}_l \in \{2h, 4h, 8h\}$.
- Each instance set uses a unique combination of $|V|, |L|, \hat{t}_l$ and contains 30 instances, resulting in a total of 1800 instances[2].

The scaling factors in the objective function were set to $\alpha^{\text{bal}} = 1$, $\alpha^{\text{load}} = \alpha^{\text{work}} = \frac{1}{10000}$. Using these factors, improving the system balance always has a greater impact on the objective value than reducing the tour lengths or the number of loading operations. The algorithm has been implemented in C++ using GCC 4.6 and each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz and 3 GB RAM per core. Each run was terminated when no improvement could be achieved within the last 5000 VNS iterations or after a CPU time of one hour. For solving the LP-based approach to determine loading instructions CPLEX 12.4 was used with default settings.

In addition to the VNS algorithm, we implemented a mixed integer programming (MIP) model similar to the sequence-indexed formulation from [2] but adapted to our problem formulation. This model is not able to consider dependencies among vehicles and is therefore restricted to the monotonic case. CPLEX 12.4 with default settings and a CPU-time limit of one hour was used for trying to solve the instances with this model. In addition, we also investigated a second MIP model based on a time-indexed formulation [2] for the general case. Experiments indicated that this approach

[1] http://www.citybikewien.at/

[2] Benchmark instances: https://www.ads.tuwien.ac.at/w/Research/Problem_Instances

Table 1. Results of the MIP approach and the VNS considering the three variants of deriving loading instructions. Each instance set contains 30 instances. All runtimes are in seconds.

Instance s			MIP			VNS with GH			VNS with MF			VNS with LP		
$\|V\|$	$\|L\|$	$\hat{t}$	$\overline{\text{ub}}$	$\overline{\text{lb}}$	$\overline{\text{t}_{\text{tot}}}$	$\overline{\text{obj}}$	sd	$\overline{\text{t}_{\text{tot}}}$	$\overline{\text{obj}}$	sd	$\overline{\text{t}_{\text{tot}}}$	$\overline{\text{obj}}$	sd	$\overline{\text{t}_{\text{tot}}}$
10	1	120	**28.3477**	28.3477	4	**28.3477**	9.9111	1	**28.3477**	9.9111	2	**28.3477**	9.9111	212
10	1	240	4.2942	0.0424	3600	**4.2941**	3.5524	5	**4.2941**	3.5524	10	**4.2941**	3.5524	1332
10	1	480	0.0320	0.0276	3600	**0.0317**	0.0033	8	**0.0317**	0.0033	17	**0.0317**	0.0033	2042
10	2	120	**9.8269**	9.4768	911	10.0266	6.3028	2	9.9601	6.2475	3	9.9600	6.2475	459
10	2	240	0.0340	0.0322	856	**0.0339**	0.0043	5	**0.0339**	0.0043	10	**0.0339**	0.0043	1441
10	2	480	**0.0317**	0.0313	1245	**0.0317**	0.0033	7	**0.0317**	0.0033	15	**0.0317**	0.0033	1797
20	2	120	55.8294	26.9012	3600	**55.0962**	13.2321	4	55.3628	13.3731	8	55.3628	13.3731	1097
20	2	240	19.7884	0.0383	3600	4.3908	3.7546	29	**4.2575**	3.7276	58	4.2576	3.7275	3600
20	2	480	1.8906	0.0403	3600	**0.0614**	0.0061	51	0.0615	0.0061	142	**0.0614**	0.0061	3600
20	3	120	37.3759	1.4777	3600	31.9096	11.9065	7	**31.7763**	11.8112	13	31.8430	11.8650	1727
20	3	240	6.2083	0.0401	3600	0.0651	0.0060	31	**0.0650**	0.0060	65	0.0652	0.0060	3600
20	3	480	13.4191	0.0316	3600	0.0616	0.0060	55	**0.0614**	0.0061	114	**0.0614**	0.0061	3600
30	2	120	106.9631	56.3908	3600	**104.7633**	17.7686	6	**104.7633**	17.7686	12	**104.7633**	17.7142	1539
30	2	240	74.9886	0.0487	3600	34.7941	10.8729	48	**34.6608**	10.4812	109	35.1940	10.9637	3600
30	2	480	69.8069	0.0432	3600	0.0926	0.0062	186	**0.0925**	0.0061	491	0.0928	0.0061	3600
30	3	120	90.4419	16.6454	3600	**78.0441**	17.2764	10	78.1773	17.0832	21	78.5771	17.2677	2521
30	3	240	61.6715	0.0461	3600	7.1526	4.7495	86	**7.1523**	4.2272	191	7.6186	4.3543	3600
30	3	480	175.4000	0.0015	3600	**0.0925**	0.0061	156	**0.0925**	0.0061	399	0.0928	0.0062	3600
60	3	120	274.3101	157.7350	3600	253.9795	27.8187	20	**253.8462**	27.6739	45	254.3794	27.3265	3600
60	3	240	370.2000	0.0000	3600	**126.7616**	20.5332	260	126.8282	20.9660	521	129.2945	20.1347	3600
60	3	480	—	—	—	**6.1766**	4.1036	1835	6.7758	4.1422	3600	10.1071	5.0800	3601
60	5	120	289.3111	34.9784	3600	197.7411	28.0192	54	**196.6749**	29.4401	99	197.0747	28.7557	3600
60	5	240	370.2000	0.0000	3600	**41.1497**	12.6579	725	41.6161	13.3489	1556	47.2145	13.0440	3600
60	5	480	—	—	—	**0.1901**	0.0090	2006	0.1902	0.0087	3600	0.1938	0.0087	3601
90	3	120	492.2319	290.8990	3600	441.5141	21.0737	35	441.6473	20.8266	82	**441.4474**	20.8250	3600
90	3	240	566.2667	0.0000	3600	295.1644	15.6493	425	**294.5646**	16.1776	985	297.3642	15.4610	3601
90	3	480	—	—	—	**100.5887**	9.6476	3600	101.1221	9.9480	3600	110.5868	9.4745	3601
90	5	120	566.2667	0.0000	3600	**375.7435**	19.5815	83	376.1432	20.6335	169	376.2767	20.5456	3600
90	5	240	—	—	—	174.9566	13.5297	1411	**174.3566**	12.7181	3304	184.8218	12.6962	3601
90	5	480	—	—	—	**1.2863**	1.5549	3600	1.6855	1.6746	3600	9.0772	3.5834	3601

unfortunately led to even worse results due to the higher complexity of the model and a required discretization of station visit times. We therefore omit these results here. Besides documenting the general suitability of the VNS and comparing it to the MIP approach, we aim at analyzing the impacts of the three alternative procedures to derive loading instructions. Table 1 lists average results for 30 instance sets (out of the 60) that appear most relevant for practice. Complete results are available for download with the benchmark instances.

For the MIP approach the table shows mean upper bounds $\overline{\text{ub}}$, mean lower bounds $\overline{\text{lb}}$, and median total run times $\overline{\text{t}_{\text{tot}}}$ for the cases where upper or lower bounds could be obtained within the time limit. The other column groups in the table show the results of the three VNS variants with GH, MF and LP applied to obtain loading instructions, respectively. For each variant mean objective values of the final solutions $\overline{\text{obj}}$, their standard deviations sd, and median total run times $\overline{\text{t}_{\text{tot}}}$ are listed. In each row best mean results are printed bold.

In general we can clearly observe that the pure MIP approach is only able to solve very small instances to optimality within the time limit. Very large gaps between lower and upper bounds show that it scales badly with increasing numbers of vehicles and especially with longer time budgets. For large instances CPLEX often only found trivial solutions where all vehicles stay at the depot, or even no solutions at all.

Among the three VNS variants, the one applying GH clearly was fastest. MF increased the running time on average by about 120%. The VNS with LP even took about 110 times longer than the VNS with MF on average for those runs that were not terminated by the time limit. Concerning solution quality, we observed that GH is able to obtain results very similar to those of MF. Both variants found better final solutions with lower objective values than the respective other variant in about 21% of runs. In the remaining 58% both approaches obtained equally good results. Objective values are on average slightly better for the MF-variant. In general, however, absolute quality differences are rather small. Also, a Wilcoxon signed-rank test does not show a significant difference regarding solution quality of GH and MF. MF runs were terminated by the time limit for the largest 8% of instances. When only comparing runs not terminated by the time limit, average objective values are more favorable for MF. However, also in this comparison the improvement over GH cannot be said to be statistically significant.

In principle the VNS with LP is sometimes able to obtain better results than the other variants since it may take advantage of not being restricted to monotonicity. Due to the substantially higher computational overhead, however, about 60% of all runs were terminated before a reasonable convergence had been achieved due to exceeding the time limit of one hour. Therefore, the LP-approach typically led to significantly worse results, particularly for larger instances. The LP-variant obtained better solutions in only 10%, while the MF-variant outperformed the LP-variant in 36% of all runs. A Wilcoxon signed-rank test confirms the assumption that the VNS with MF performs better w.r.t. solution quality with a very low error probability of less than 0.01%.

Figure 3 shows typical relative success rates for the VND neighborhoods on a large instance. In the VNS, all shaking neighborhoods have similar relative success rates, therefore we omit the corresponding chart. These results show that all neighborhood structures contribute well to the overall performance.

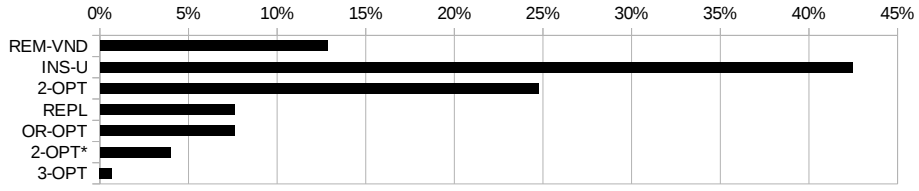

Fig. 3. Relative success rates of VND neighborhoods for an instance with $|V| = 90$, $|L| = 5$, $\hat{t} = 480$ using the MF-variant.

7 Conclusions and Future Work

We presented a VNS metaheuristic with an embedded VND for solving the balancing bicycle sharing system problem. Main ingredients are a meaningful greedy construction heuristic for generating initial solutions, neighborhood structures derived from VRPs, new problem-specific neighborhood structures, as well as three alternatives for deriving optimized loading instructions for created candidate tours. Experimental results on instances derived from real-world data show that the VNS in general performs well and scales much better than two MIP approaches. Concerning the derivation of loading instructions, the greedy method is fastest and delivers solutions similar in quality to those of the more complex maximum flow based approach. The LP-based method has the advantage of being able to find optimal loading instructions even for the general, not necessarily monotonic case, but unfortunately the added flexibility cannot compensate the typically much larger computational effort when considering reasonable runtime limits. Thus, the fast greedy method is the best compromise for practice.

In future work, we intend to model the times needed for loading bikes at a station more accurately by taking the number of loading actions into account instead of assuming average dwell times. Another practically relevant extension is to allow vehicles to start and return nonempty. Finally, we also want to turn towards the dynamic scenario, where the fill levels at stations change during the balancing process. Stochastic aspects then also need to be considered. Last but not least, hybridizing the VNS with the MIP approaches, e.g., by including some MIP-based large neighborhood search, appears to be promising.

References

1. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. To appear in Discrete Optimization (2012)
2. Raviv, T., Tzur, M., Forma, I.A.: Static Repositioning in a Bike-Sharing System: Models and Solution Approaches. To appear in EURO Journal on Transportation and Logistics (2012)
3. Benchimol, M., Benchimol, P., Chappert, B., De la Taille, A., Laroche, F., Meunier, F., Robinet, L.: Balancing the stations of a self service bike hire system. RAIRO – Operations Research **45**(1) (2011) 37–61
4. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Technical Report CIRRELT-2012-09, CIRRELT, Montreal, Canada (2012) submitted to Transportation Science.
5. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(11) (1997) 1097–1100
6. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica **19**(4) (1997) 390–410
7. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In Prodhon, C., et al., eds.: Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France (2008)
8. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers (2003) 219–249