Balancing Bicycle Sharing Systems: Improving a VNS by Efficiently Determining Optimal Loading Operations

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

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

Abstract. Public bike sharing systems are important alternatives to motorized individual traffic and are gaining popularity in larger cities worldwide. In order to maintain user satisfaction, operators need to actively rebalance the systems so that there are enough bikes available for rental as well as sufficient free slots for returning them at each station. This is done by a vehicle fleet that moves bikes among the stations. In a previous work we presented a variable neighborhood search metaheuristic for finding effective vehicle routes and three different auxiliary procedures to calculate loading operations for each candidate solution. For the most flexible auxiliary procedure based on LP, the current work provides a new, practically more efficient method for calculating proven optimal loading operations based on two maximum flow computations. The different strategies for determining loading operations are further applied in combination controlled by an additional neighborhood structure. Experimental results indicate that this combined approach yields significantly better results than the original variable neighborhood search.

1 Introduction

Public bicycle sharing systems are booming worldwide in many major cities as they augment public transport very well [1, 2]. Modern systems have automated rental stations where users can easily rent bikes and return them elsewhere. In order to achieve a high degree of acceptance, operators need to actively rebalance the system in order to ensure that there are enough bikes as well as parking slots for returning them at any station at almost all times. This balancing is typically done by a vehicle fleet with trailers. So far, drivers mostly follow their experience and intuition when planning the transportation routes. It was not until recently that researchers have started to consider this transportation planning problem from an optimization point of view.

The *Balancing Bicycle Sharing System* (BBSS) problem is related to the well-studied vehicle routing problem (VRP). However, there are significant differences such as allowing multiple visits at stations and that an arbitrary number of bikes may be loaded or unloaded at each visit. We consider the static variant of the BBSS problem in which user activities during the rebalancing process are neglected. It can be regarded as a capacitated single commodity split pickup and delivery VRP. So far, only few algorithms have been published for the BBSS problem, and due to specific application characteristics, they address significantly different variants.

Chemla et al. [2] consider the problem with only one vehicle and achieving perfect balance as hard constraint. They describe a branch-and-cut algorithm utilizing a relaxed mixed integer linear programming (MIP) model and a tabu search for locally improving incumbent solutions. Benchimol et al. [3] also assume balancing as hard constraint and focus on approximation algorithms for selected special situations. Raviv et al. [4] propose four MIP models for different problem variants and compare their assets and drawbacks on instances with up to 60 stations. Their objective function minimizes user dissatisfaction and ignores tour lengths as well as the number of loading operations.

Contardo et al. [5] consider different MIP models for the dynamic scenario where demands need to be satisfied over time. They propose a hybrid approach using column generation and Benders decomposition that is able to handle instances with up to 100 stations.

Schuijbroek et al. [6] describe the decomposition of the problem into separate single-vehicle routing problems by solving a polynomial-size clustering problem. They apply a clustered MIP heuristic in two versions, with and without additional cuts. In addition, they present a constraint programming model that represents the problem as a scheduling problem. Results on instances of up to 135 stations and five vehicles show that the approaches outperform a MIP model operating on the full unclustered problem.

In [7], we propose a variable neighborhood search (VNS) metaheuristic for finding effective vehicle routes that employs an auxiliary algorithm for calculating meaningful loading operations for each considered candidate set of routes. Three alternatives have been studied for this auxiliary algorithm, with the most precise but also slowest one being based on linear programming (LP). While the first two methods are restricted to the so-called monotonic case, where stations may not be used as temporary buffers for the redistribution of bikes, the LP-approach is more flexible.

The current work improves upon these methods by introducing a practically significantly more efficient method for determining proven optimal loading operations for the general case based on two maximum flow computations. We also investigate a unified approach where multiple strategies for determining loading instructions are applied in combination. This is achieved by an additional neighborhood structure that determines the best suited strategy. Computational tests are performed on instances derived from real-world scenarios, indicating that the unified approach performs significantly better. However, it is also shown that in most cases the quality-loss of restricting the algorithm to monotonicity, i.e., not allowing buffering, is only small.

2 Problem Definition

Formally the BBSS problem is defined on a complete directed graph $G_0 = (V_0, A_0)$, where node set $V_0 = V \cup \{0\}$ consists of nodes for the rental stations V plus the vehicles' depot 0. Each arc $(u, v) \in A_0$ has associated a travel time $t_{u,v} > 0$ that includes a surcharge for parking and loading/unloading bikes. By $G = (V, A),\ A \subset A_0$ we denote the subgraph induced by the stations V only.

Each station $v \in V$ has associated a capacity of bikes $C_v \geq 0$, i.e., the number of available parking positions, the number of bikes it initially contains $p_v \geq 0$, and a target number of bikes it should contain 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 bikes $Z_l > 0$ and starts and ends its route at the depot 0.

When serving a station, we must not only consider the traveling time given by the corresponding arc, but also the duration needed for parking the vehicle at the station and for performing loading operations. We can simply add the parking time and the time needed for an average number of loading operations to the traveling time, and therefore do not need to handle them separately anymore. In the following sections it is assumed that the traveling times have already been preprocessed in this sense.

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. Stations may be visited multiple times by the same or different vehicles. The total time t_l of a route may never exceed a given time limit $\hat{t}$. As the start and end point of each tour is the depot 0, it is not explicitly stored but assumed to be prepended and appended, respectively. The second part are the loading instructions $y_{l,v}^i, \in \{-Z_l, \ldots, Z_l\}$ with $l \in L,\ v \in V$, and $i = 1, \ldots, \rho_l$, specifying how many bikes are picked up if $y_{l,v}^i > 0$ or delivered if $y_{l,v}^i < 0$, respectively, at vehicle l's i-th stop at station v.

The following conditions must hold: The number of bikes available at each station $v \in V$ never exceeds C_v, for any vehicle $l \in L$ its capacity Z_l may never be exceeded, and the total time t_l of a tour

$$t_l = t_{0,r_l^1} + \sum_{i=2}^{\rho_l} t_{r_l^{i-1},r_l^i} + t_{r_l^{\rho_l},0} \tag{1}$$

may not exceed the time limit $\hat{t}$.

Let a_v be the final number of bikes at each station $v \in V$ after rebalancing.

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

The primary objective is to minimize the deviation from the target number of bikes $\delta_v = |a_v - q_v|$ at each station $v \in V$, i.e., its *disbalance*, and secondarily we aim at minimizing the number of loading/unloading operations as well as the overall time required for all routes, i.e.,

$$\min \quad \alpha^{\text{bal}} \sum_{v \in V} \delta_v + \alpha^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_{l,r^i_l}| + \alpha^{\text{work}} \sum_{l \in L} t_l, \tag{3}$$

where $\alpha^{\text{bal}} \gg \alpha^{\text{load}}, \alpha^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms. Note that an improvement in the balance is always considered as better than any improvements in the secondary objectives; throughout this paper we use $\alpha^{\text{bal}} = 1,\ \alpha^{\text{load}} = \alpha^{\text{work}} = 1/100\,000$. Despite this small weight, in case of equal balance the secondary objectives become important criteria to distinguish solutions and therefore must not be neglected. Otherwise, e.g. obviously unnecessary stops or loading and unloading actions may occur.

A natural simplification that may be exploited is to consider *monotonicity* regarding the 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 bikes at pickup stations and unload them at delivery stations. Depending on the excess or shortage of bikes at a station, vehicles are only allowed to load or unload bikes at it, respectively. In this way the number of bikes decreases or increases monotonically, and consequently the order in which different vehicles visit a station does not matter. Monotonicity simplifies the task of finding optimal loading instructions for a given set of routes considerably [7]. On the downside, enforcing monotonicity may exclude better solutions that e.g. use stations as buffers to temporarily store bikes or to transfer bikes between vehicles, see Fig. 1.

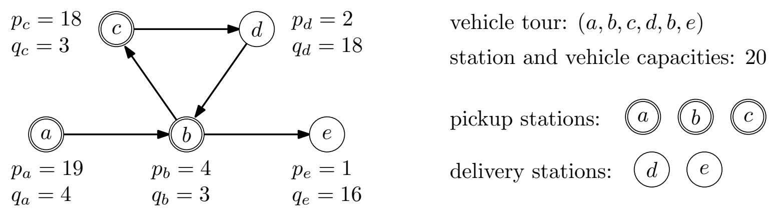
$p_c = 18$
$q_c = 3$
c
d
$p_d = 2$
$q_d = 18$
a
b
e
$p_a = 19$
$q_a = 4$
$p_b = 4$
$q_b = 3$
$p_e = 1$
$q_e = 16$
vehicle tour: (a, b, c, d, b, e)
station and vehicle capacities: 20
pickup stations: a b c
delivery stations: d e

Fig. 1. Example where the restriction to monotonicity yields a worse solution. With monotonicity, the best possible loading instructions are $y_1 = (+15, +1, +4, -16, 0, -4)$ resulting in a total disbalance of 22. In the general case, node b can be used as buffer and loading instructions $y_1 = (+15, -14, +15, -16, +15, -15)$ yield perfect balance.

3 Variable Neighborhood Search for BBSS

Our metaheuristic for BBSS follows the general variable neighborhood search (VNS) principle [8] and is described in detail in [7]. An embedded Variable Neighborhood Descent (VND) is used for deterministic local improvement as intensification, while the outer VNS relies on stochastic shaking in larger neighborhoods for diversification.

An initial solution is derived by a greedy heuristic that iteratively constructs vehicle routes by always appending a feasible station for which the possible gain in balance divided by the additional travel time is maximal. As vehicles have to terminate their routes empty, special attention is paid when considering pickup-stations: It is estimated how many bikes can still be delivered after the potential visit of a pickup-station and the possible balance gain is adjusted accordingly, i.e., the number of bikes that may be picked up is restricted correspondingly. Thus, loading instructions are set in this construction heuristic in a purely greedy way, i.e., by always picking up or delivering as many bikes such that the balance gain is locally maximized at each station visit.

In contrast, the VNS with its embedded VND searches the space of vehicle routes exploiting eleven different neighborhood structures only, and corresponding loading instructions are always derived for each considered set of routes by an auxiliary algorithm. Three alternatives have been investigated in [7]: a greedy heuristic (GH), a maximum flow approach for the monotonic case (MF-MC), and a linear programming approach for the general case (LP).

GH considers the stations in the order as they are visited in each tour and tries to bring each station as far as possible towards balance. Due to its greedy nature, the derived loading instructions do not necessarily provide the best possible overall balance and/or minimal number of loading/unloading activities. MF-MC sets up a flow network according to the given routes. By computing a maximum flow on this network, it is possible to obtain optimal loading instructions yielding the lowest achievable imbalance and a minimal number of loading operations under the assumption of monotonicity. While GH is fastest, MF-MC still is computationally very efficient, taking only about 1.8 times longer on average in our experimental evaluation. In order to overcome the monotonicity restriction, LP solves a minimum cost flow problem on a more sophisticated network via linear programming. On average our implementation of LP with CPLEX 12.4 needs about 90 times longer than MF-MC. This huge disadvantage, unfortunately, implies that the VNS can only perform substantially less iterations, and this aspect cannot be compensated by the higher quality of the loading instructions. In Section 4 we will present a new, computationally significantly more efficient approach to obtain optimal loading instructions for the general case.

The following subsections summarize the VND/VNS neighborhood structures. We employ several classical neighborhood structures that were already successfully applied in various VRPs together with new structures exploiting specifics of BBSS. Concerning the classical neighborhood structures, we primarily based our design on the experience from [9].

3.1 VND Neighborhood Structures

The following neighborhoods are applied in the given order and searched in a best improvement fashion. Preliminary experiments with a dynamic reordering strategy did not yield any significant advantage. All considered candidate tours are checked for feasibility, infeasible solutions are discarded. For each feasible solution one of the above mentioned methods for deriving loading instructions is applied. Obsolete station visits where no loading actions performed are removed from the tours.

Remove station: Considers all single station removals to avoid unnecessary visits.

Insert unbalanced station: Considers the insertion of any yet unbalanced station at any possible position.

Intra-route 2-opt: The classical 2-opt neighborhood of the traveling salesman problem applied individually to each route.

Replace station: Considers the replacement of each single station by another yet unbalanced station.

Intra or-opt: Considers all solutions in which sequences of one, two, or three consecutive stations are moved to a different place within the same route.

2-opt* inter-route exchange: Considers all feasible exchanges of arbitrarily long end segments of two routes.

Intra-route 3-opt: 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.

3.2 VNS Neighborhoods Structures

Shaking selects solutions randomly from the following neighborhood, which are all parameterized by δ, yielding a total of 24 specific neighborhoods. In contrast to the VND, created routes that violate the time budget are repaired by removing stations from the end. The neighborhoods are again applied in the given order, and each derived candidate solution is locally improved by the VND before deciding upon its acceptance.

Move sequence: 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: 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: 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 up 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 [10] with the threshold parameter set to $\delta \in \{0\%, 4\%, 8\%, 12\%, 16\%, 20\%\}$.

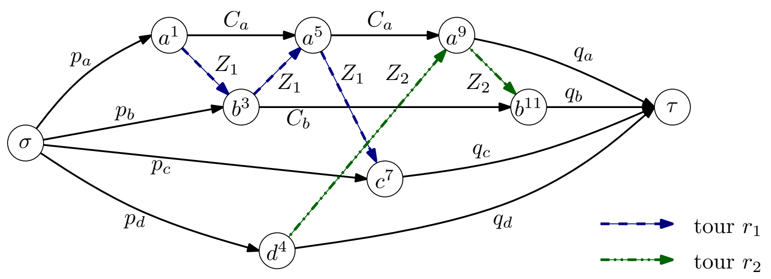

Fig. 2. Exemplary flow network for vehicle routes $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$.

4 Maximum Flow Based Method for the General Case

Similarly as in [2, 7], we set up a flow network, which is illustrated in Fig. 2. By $t(r_l^i)$ we denote the time when vehicle l makes its i-th stop at station r_l^i. Let $G_{\text{f}} = (V_{\text{f}}, A_{\text{f}})$ be a directed multi-graph with node set $V_{\text{f}} = \{\sigma, \tau\} \cup V_t$, where σ and τ are source and target nodes, respectively, and $V_t = \{v^j \mid v = r_l^i,\ j = t(r_l^i),\ i = 1, \dots, \rho_l,\ l \in L\}$; i.e., we have a node v^j for each station v and time j when some vehicle makes a stop at v. Let $V^{\text{first}} = \{v^{j_{\min}} \in V_t \mid j_{\min} = \min\{j \mid v^j \in V_t\}\}$, i.e., the nodes representing 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., the nodes representing 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 $A_{R,l} = \{(u^j, v^k) \mid u = r_l^i,\ v = r_l^{i+1},\ j = t(r_l^i),\ k = t(r_l^{i+1}),\ i = 1, \dots, \rho_l - 1\},\ \forall l \in L$, i.e., 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 time.
- $A_V = \bigcup_{v \in V} A_v,\ A_v = \{(v^{j_1}, v^{j_2}), \dots, (v^{j_{\max-1}}, v^{j_{\max}})\}$ with $(v^{j_1}, \dots, v^{j_{\max}})$ being the sequence of nodes representing visits of station v sorted according to time. These arcs model the bikes staying at a station, capacities are C_v.

Step 1 – minimizing disbalance: In the first step, we calculate the maximum (σ, τ)-flow on this network. As argued in [2, 7] the value of this maximum flow corresponds to the maximum achievable reduction of disbalance. In the ideal case all arcs $A_\sigma \cup A_\tau$ are fully saturated in the solution, indicating that the target values q_v can be achieved at all visited stations v. Loading instructions $y_{l,v}^i$ are obtained by taking the flow differences among successive arcs $(u^j, v^k) \in A_R$ for each vehicle and each stop.

However, these loading instructions may be infeasible if an arc $(\sigma, v^j) \in A_\sigma$ is not saturated. In this case, there are actually more bikes at station v than assumed in the flow network, and delivering bikes to this station may exceed the station's capacity.

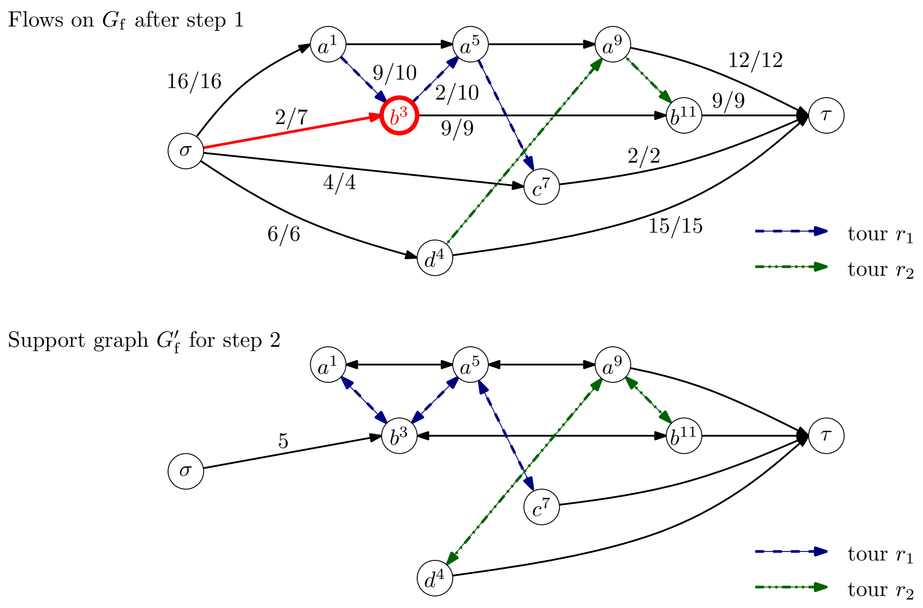

Fig. 3. Example where the arc (σ, b^3) is not saturated after the first maximum flow computation and therefore the solution is infeasible. In step 2 the remaining commodities of (σ, b^3) must be rooted through G'_f. (Flow and capacity values on several arcs are omitted for better readability.)

Step 2 – saturating arcs A_σ: To repair the above situation, we perform a second stage maximum flow computation, modifying the loading instructions to become feasible while the total imbalance remains unchanged. The basic idea is to increase the capacities of the A_τ arcs and push additional flow through the network in order to saturate all arcs A_σ. Let $f(u^j, v^k)$ denote the flow on an arc (u^j, v^k). We derive from $G_\text{f} = (V_\text{f}, A_\text{f})$ a support graph $G'_\text{f} = (V_\text{f}, A'_\text{f})$ with the same node set but a modified arc set $A'_\text{f} = A'_\sigma \cup A'_\tau \cup A'_R \cup A'_V$ with:

- $A'_\sigma = \{(\sigma, v^j) \in A_\sigma \mid f(\sigma, v^j) < p_v\}$ with capacities $C_v - f(\sigma, v^j)$.
- $A'_\tau = \{(v^j, \tau) \in A_\tau\}$ with capacities $C_v - f(v^j, \tau)$.
- A'_R contains arcs $(u^j, v^k) \in A_R$ with residual capacities $Z_l - f(u^j, v^k)$ and corresponding reverse arcs (v^k, u^j) with capacities $f(u^j, v^k)$.
- A'_V contains arcs $(v^j, v^k) \in A_V$ with residual capacities $C_v - f(v^j, v^k)$ and corresponding reverse arcs (v^k, v^j) with capacities $f(v^j, v^k)$.

Subsequently, we perform a second maximum flow computation on G'_f, which always saturates every arc in A'_σ since $p_v \leq C_v,\ \forall v \in V$ holds. By modifying the original flows on A_R with the new flows $f'(u^j, v^k)$ on $(u^j, v^k) \in A'_R$, we obtain corrected flows $f_\text{corr}(u^j, v^k) = f(u^j, v^k) + f'(u^j, v^k) - f'(v^k, u^j),\ \forall (u^j, v^k) \in A_R$. Loading instructions derived from these flows are feasible and optimal with respect to the achievable balance. Figure 3 shows an example of G'_f.

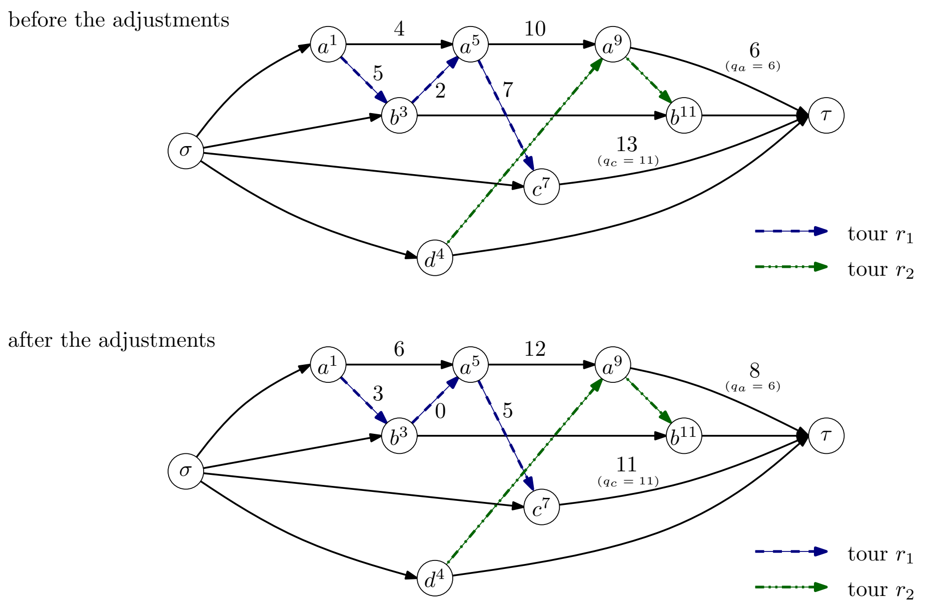

Fig. 4. Example of how the flow values can be adjusted in order to decrease the number of loading operations by 8.

Step 3 – minimizing the number of loading operations: While we cannot do better with respect to balance, we so far neglected the second term of the objective function (3) which aims at avoiding unnecessary loading operations. In order to further optimize this aspect, we adjust the corrected flow values f_{corr} on G_{f} in a way that we do not change the overall flow, but the load is shifted from transportation arcs A_R (which influence the loading instructions) to arcs A_V and A_τ. We aim at reaching the final number of bikes at each station as early as possible so that the number of loading operations is minimized. The algorithm works as follows:

(1) Consider all nodes in V_t according to the visiting times:
(2) The next node $v^j \in V_t$ either has an outgoing arc $a = (v^j, \tau) \in A_\tau$ or $a = (v^j, v^k) \in A_V,\ j < k$.
(3) Find a path P in G_{f} from v^j that does not contain arc a and either ends in a node $v^k,\ j < k$ or in node τ. If no such path exists, continue at (2).
(4) Let P' be the alternate path from v^j to the end node of P starting with a and otherwise including only arcs from $A_v \cup A_\tau$. The goal is to adjust the flows f_{corr} on all arcs in P and P' by a common value Δ where
   - flow values on arcs of P are decreased by Δ and must not become negative or less than q_v, and
   - flow values on arcs of P' are increased by Δ and may not exceed their capacity limits.

If P and P' do not end at τ, we use Δ directly as the adjustment value. If the paths end at τ, the situation becomes more complicated since adjustments modify the balance of some stations. Let (u^i, τ) be the last arc in P and (v^j, τ) the last arc in P'. Let $bal(P) = f_{\text{corr}}(u^i, \tau) - q_u$ and $bal(P') = f_{\text{corr}}(v^j, \tau) - q_v$. We have to consider the following four cases.
    (a) If $bal(P) \geq 0$ and $bal(P') \geq 0$, adjust by $\min(\Delta, bal(P))$.
    (b) If $bal(P) \geq 0$ and $bal(P') < 0$, adjust by $\min(\Delta, \max(bal(P), bal(P')))$.
    (c) If $bal(P) < 0$ and $bal(P') \geq 0$, do not adjust.
    (d) If $bal(P) < 0$ and $bal(P') < 0$, adjust by $\min(\Delta, bal(P'))$.
(5) Repeat (3) and (4) until no further adjustments are possible. If any adjustments were found, restart the algorithm from (1), else continue at (2).

An example of this procedure is given in Fig. 4. First the arc (a^1, a^5) is considered and we can find improvements on paths $P = \langle a^1, b^3, a^5 \rangle$ and $P' = \langle a^1, a^5 \rangle$. Increasing the flow on arcs of P' and decreasing the flow on arcs of P by 2 does not change the final balance of any stations, but reduces the number of loading operations by 4. The second improvement can be found on arc (a^5, a^9) and paths $P = \langle a^5, c^7, \tau \rangle$ and $P' = \langle a^5, a^9, \tau \rangle$. Increasing the flow on arcs of P' and decreasing the flow on arcs of P by 2 now changes the balance at stations a and c. However, shifting the two excessive bikes from c to a is cost-neutral in terms of final balance, but reduces the number of loading operations by 4.

Note that the restart at step (5) is necessary since adjustments on paths from a station v with later visiting times may have an impact on paths from an earlier station u by enabling adjustments that were not possible when u was considered. It can be shown that loading instructions derived from the resulting flows after this procedure are optimal with respect to balance and the number of loading operations as long as the route is local optimal with respect to the VND neighborhoods, i.e., it does not contain any unnecessary stops that can be removed without increasing the disbalance.

5 Combined Approach

Experiments with the new maximum flow based method for the general case indicated a substantial speedup in comparison to the LP-based method. Nevertheless, also this strategy is still significantly slower than the simple greedy method. Experiments shown in the next section indicate that it remains questionable whether the advantage of potentially better solutions outweighs the disadvantage of higher running times. We therefore also investigated algorithms, where the individual strategies for determining loading instructions (except the LP-based method which is now clearly dominated) are applied in combination.

The following approach turned out to work particularly well: We start with the fast greedy method as default strategy and introduce an additional VND neighborhood structure, inserted at the fourth position: All available strategies for determining loading instructions are applied and their results are compared. The best strategy, which is the one yielding the best solution or in case of ties the fastest one, is remembered and from now on also applied as default strategy for all successively created candidate solutions until the solution for which this strategy was found best is discarded in a VNS iteration due to a better non-descending solution. In this latter case, the default strategy is reset to the fast greedy method.

6 Computational Results

We tested our approach on benchmark instances[1] from [7]. These instances are based on real-world data provided by Citybike Wien[2] running a bike-sharing system with currently 92 stations. The instances are characterized by the number of stations $|V| \in \{10, 20, 30, 60, 90\}$, the number of vehicles $|L| \in \{1, 2, 3, 5\}$, and the shift length $\hat{t} \in \{120, 240, 480\}$. 30 instances are considered for each combination of a subset of practically meaningful configurations. The algorithms have been implemented using GCC 4.6 and each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz. Each run was terminated when no improvement could be achieved within the last 5 000 VNS iterations or after one hour of CPU time. In the first case we consider the heuristic search as converged, major further improvements would be highly unlikely. For all maximum flow calculations we use the push-relabel method by Cherkassky and Goldberg [11], while CPLEX 12.4 is used in the LP approach.

Table 1 shows average results of our VNS with the five methods for deriving loading instructions: GH, MF-MC, and LP from [7] and the two new methods denoted by MF-GC (for "maximum flow based, general case") and COMB (for "combined approach"). For each algorithm variant and each instance class we list the number of instances (runs) for which the algorithm variant yielded the best results (#best), the mean objective value of the finally best solutions ($\overline{\text{obj}}$), the corresponding standard deviation (sd), and the median run time until the best solutions have been found ($\widetilde{\text{time}}$). Note that objective values must be compared with care due to the small scaling factor for the secondary objectives ($\alpha = 1/100\,000$). In case of two solutions achieving the same balance, objective value differences may be very small, but they might nevertheless indicate practically important differences in the lengths of routes or the number of loading instructions. Thus, we consider #best to be a better indicator for analyzing performance differences than objective value differences. Maximum #best-values are printed bold for each instance class. Note that in comparison to the results published in [7], all former methods exhibit slightly better average objective values due to some small but significant improvements in the implementation.

As one might expect, we observe that in general GH is clearly the fastest variant while LP is slowest. On average over the larger instances for which the runs were terminated by the time limit of one hour, GH could perform 110 times more iterations than LP. MF-MC increased the running time over GH per iteration on average by about 120% and MF-GC by about 290%. They are thus substantially faster than LP but still considerably slower than GH. In

[1] available at https://www.ads.tuwien.ac.at/w/Research/Problem_Instances

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

Table 1. Results of the VNS with different variants for deriving loading instructions.

Inst. set			**VNS/GH**				**VNS/MF-MC**				**VNS/LP**				**VNS/MF-GC**				**VNS/COMB**			
$\|V\|$	$\|L\|$	$\hat{t}$	#best	$\overline{\text{obj}}$	sd	$\overline{\text{time}}$	#best	$\overline{\text{obj}}$	sd	$\overline{\text{time}}$	#best	$\overline{\text{obj}}$	sd	$\overline{\text{time}}$	#best	$\overline{\text{obj}}$	sd	$\overline{\text{time}}$	#best	$\overline{\text{obj}}$	sd	$\overline{\text{time}}$
10	1	120	28	28.334	9.911	0.0	29	28.334	9.911	0.0	29	28.334	9.911	0.1	**30**	28.334	9.911	0.0	29	28.334	9.911	0.0
10	1	240	28	4.269	3.551	0.0	29	4.269	3.551	0.0	29	4.269	3.551	1.2	**30**	4.202	3.575	0.0	28	4.269	3.551	0.0
10	1	480	26	0.003	0.000	0.0	**29**	0.003	0.000	0.0	**29**	0.003	0.000	1.2	**29**	0.003	0.000	0.0	27	0.003	0.000	0.0
10	2	120	27	10.002	6.302	0.0	27	10.002	6.302	0.0	**30**	9.936	6.247	0.4	29	9.936	6.247	0.0	28	10.069	6.356	0.0
10	2	240	**30**	0.003	0.000	0.0	**30**	0.003	0.000	0.0	29	0.003	0.000	4.9	**30**	0.003	0.000	0.1	28	0.003	0.000	0.0
10	2	480	26	0.003	0.000	0.0	**29**	0.003	0.000	0.0	28	0.003	0.000	1.2	28	0.003	0.000	0.0	27	0.003	0.000	0.0
20	1	120	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.1	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.0
20	1	240	28	43.003	11.335	0.0	28	43.003	11.383	0.0	**29**	42.936	11.298	6.5	**29**	43.003	11.335	0.2	**29**	43.003	11.335	0.0
20	1	480	**29**	1.672	2.171	0.7	27	1.672	2.171	1.7	26	1.805	2.187	155.3	27	1.672	2.171	12.7	25	1.672	2.170	0.6
20	2	120	28	55.336	13.373	0.0	28	55.336	13.373	0.0	29	55.336	13.373	2.8	**30**	55.336	13.373	0.1	**30**	55.336	13.373	0.0
20	2	240	20	4.139	3.748	3.2	**22**	4.205	3.726	3.3	16	4.205	3.726	237.3	**22**	4.272	3.704	17.4	18	4.205	3.800	1.3
20	2	480	22	0.006	0.000	2.1	25	0.006	0.000	5.2	20	0.006	0.000	240.9	24	0.006	0.000	7.9	**26**	0.006	0.000	2.0
30	2	120	24	104.802	17.877	0.0	26	104.802	17.877	0.0	**27**	104.736	17.768	7.9	26	104.802	17.792	0.2	26	104.603	17.625	0.0
30	2	240	15	34.472	10.682	4.8	10	34.672	10.781	10.4	13	35.006	10.735	1155.6	**18**	34.739	10.808	58.8	15	34.539	10.966	3.7
30	2	480	16	0.009	0.000	26.1	17	0.009	0.000	36.6	5	0.009	0.000	1490.6	16	0.009	0.000	240.6	**19**	0.009	0.000	31.6
30	3	120	21	78.204	17.343	0.1	27	77.737	17.065	0.5	**23**	78.204	17.438	53.4	**23**	78.004	17.356	1.7	**23**	77.937	17.280	0.3
30	3	240	**11**	7.208	4.475	28.1	7	7.141	4.539	38.4	1	7.675	4.551	1808.4	**11**	6.875	4.384	323.2	7	7.008	4.259	17.9
30	3	480	13	0.009	0.000	32.9	13	0.009	0.000	42.5	4	0.009	0.000	1479.0	**16**	0.009	0.000	392.3	**16**	0.009	0.000	46.3
60	1	120	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.7	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.0
60	1	240	26	267.336	23.811	0.5	24	267.336	23.794	0.6	26	267.336	24.276	77.6	**28**	267.136	23.842	1.8	26	267.203	23.766	0.4
60	2	480	4	60.412	13.629	309.1	7	60.212	13.889	1006.1	0	63.745	13.803	2969.3	5	61.412	14.202	2094.8	**14**	59.879	13.845	474.1
60	3	240	8	126.742	20.003	68.2	**11**	126.942	20.475	129.8	2	129.209	20.828	2946.9	8	126.876	19.928	593.5	8	127.342	20.153	65.5
60	3	480	8	6.284	3.921	906.2	7	6.350	4.071	2508.3	0	10.617	5.096	2856.3	2	6.884	3.471	2648.4	**15**	5.950	3.805	990.6
60	5	120	8	196.940	28.665	6.6	13	196.474	29.203	12.9	6	197.074	29.005	865.1	13	196.474	29.147	97.2	**18**	196.207	29.390	9.2
60	5	240	7	41.814	13.568	408.2	7	41.548	13.304	720.8	0	46.814	13.293	3085.9	4	42.348	12.251	2342.3	**14**	41.148	12.710	427.2
90	1	120	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.4	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.0
90	1	240	**29**	456.936	21.125	0.8	28	456.936	21.281	2.8	**29**	456.870	21.209	110.2	**29**	456.936	21.125	6.7	**29**	456.936	21.125	0.9
90	1	480	15	352.540	17.702	17.3	14	352.673	18.280	102.4	7	353.740	18.365	815.3	**16**	352.673	18.317	328.7	**16**	352.673	18.021	35.8
90	2	120	25	478.736	21.987	0.3	23	478.936	21.818	0.6	25	478.869	21.900	70.2	**27**	478.803	21.881	2.0	**27**	478.803	21.975	0.3
90	2	240	13	368.140	18.479	32.0	**14**	368.206	18.105	83.7	7	369.406	18.187	1708.2	13	368.273	18.343	209.6	13	368.540	18.292	37.0
90	2	480	**11**	205.279	13.250	493.1	6	205.013	13.500	1733.8	0	210.879	14.070	2987.4	4	206.679	13.102	2663.3	9	205.146	12.819	875.2
90	3	120	23	441.938	20.946	1.2	21	441.871	20.724	3.3	23	441.804	21.362	221.2	**25**	441.604	20.647	11.4	23	441.738	20.882	2.9
90	3	240	9	294.809	16.320	99.6	9	293.609	16.020	304.4	0	297.609	15.361	2598.0	**12**	294.276	16.109	1133.2	6	294.476	15.971	198.0
90	3	480	11	100.352	9.571	2007.1	6	101.218	10.300	2526.2	0	110.485	9.919	2364.8	0	104.418	9.970	2759.2	**13**	99.952	9.119	2320.3
90	5	120	5	376.141	20.109	13.3	9	376.140	19.874	23.2	10	376.141	19.333	1631.5	**12**	375.807	20.405	161.7	10	375.741	20.299	19.2
90	5	240	**10**	174.282	13.541	594.7	**10**	174.415	12.653	1451.5	0	185.482	13.108	3040.4	3	177.348	13.007	2479.2	9	174.149	13.472	769.2
90	5	480	**15**	0.961	1.638	2888.7	2	1.361	1.688	3018.2	0	8.494	3.223	2657.8	0	3.028	2.149	3091.2	14	1.028	1.364	3139.7
TOTAL			709	5252.403		7944.9	704	5251.737		13767.2	592	5308.335		38654.0	709	5263.486		21679.4	**755**	**5249.203**		9469.2

contrast, the combined strategy increases the average runtime only moderately by about 20%. Thus, the number of iterations COMB could perform for the larger instances where it has been terminated by the run time limit was not dramatically lower than the iteration number of GH.

Concerning the total number of instances on which an algorithm performed best (sum of all #best) COMB is the clear winner with 755 instances, followed by GH and MF-GC with the same value of 709 instances, and closely behind MF-MC with 704 instances. Final objective values give a similar picture although observed differences are generally very small. Wilcoxon signed-rank tests indicate the statistical significance of the superiority of COMB over all other approaches and that LP is dominated by all other approaches with error probabilities of less than 3%.

It is relatively hard to derive more detailed conclusions of the performances w.r.t. the parameters $|V|$, $|L|$, and $\hat{t}$, but we can observe the general tendency that LP can only compete on small instances, while GH, MF-MC, MF-GC, and COMB perform comparably well over the whole range of benchmark instances.

7 Conclusions and Future Work

We presented a new method for calculating proven optimal loading instructions for given routes to be used within a heuristic VNS framework for obtaining approximate solutions for the static problem of balancing a bicycle sharing system. This method, called MF-GC, is not restricted by monotonicity and is based on two sequential maximum-flow calculations and a final phase for minimizing the number of loading operations. The experiments performed have shown that this new method is in practice substantially faster than LP. The VNS with MF-GC is therefore able to perform much more iterations than with LP in reasonable time, and consequently significantly better final solutions are obtained. Nevertheless, the greedy heuristic calculation of loading instructions is still a very strong competitor, since it is even faster and yields results that are typically very close to optimal, despite the fact that it produces only monotonic solutions. These results also show, that only infrequently significant advantages can be gained by exploiting non-monotonicity, i.e., by using stations as buffers or transferring bikes from one vehicle to another.

Finally, we applied the strategies GH, MF-MC, and MF-GC in combination, controlled by an additionally introduced VND-neighborhood structure. This approach turned out to exploit the benefits of the individual strategies in a beneficial way – it applies the more expensive MF-MC and MF-GC only on a more regular basis when an advantage could already be gained in a predecessor solution. This approach performed best with high statistical significance, although objective value differences still remain small.

In future work we want to investigate further large neighborhood structures that might be based on ejection chains or mixed integer programming. Furthermore the underlying problem definition needs to be extended to cover the dynamic case, in which bikes rented and brought back during the rebalancing process are also considered. As these demands are not exactly known in advance, this extended problem variant becomes a stochastic online problem.

Acknowledgements

This work is supported by the Austrian Research Promotion Agency (FFG) under contract 831740. We thank Matthias Prandtstetter, Andrea Rendl, and Markus Straub from the Austrian Institute of Technology (AIT) and City Bike Wien for the collaboration in this project, constructive comments and for providing the data used in our test instances.

References

1. DeMaio, P.: Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation **12**(4) (2009) 41–56
2. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization **10**(2) (2013) 120–146
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. Raviv, T., Tzur, M., Forma, I.A.: Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transportation and Logistics (2013) 1–43
5. 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.
6. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Technical Report 2013-E1, Tepper School of Business, Carnegie Mellon University (2013)
7. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In Middendorf, M., Blum, C., eds.: Evolutionary Computation in Combinatorial Optimization. Volume 7832 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2013) 121–132
8. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(11) (1997) 1097–1100
9. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In Prodhon, C., Wolfler-Calvo, R., Labadi, N., Prins, C., eds.: Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France (2008)
10. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Kluwer Academic Publishers (2003) 219–249
11. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica **19**(4) (1997) 390–410