TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

# PILOT, GRASP, and VNS Approaches for the Static Balancing of Bicycle Sharing Systems

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

Forschungsbericht / Technical Report

**TR–186–1–14–01**

January, 21 2014

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

**Noname manuscript No.**
(will be inserted by the editor)

# PILOT, GRASP, and VNS Approaches for the Static Balancing of Bicycle Sharing Systems

**Marian Rainer-Harbach · Petrina Papazek · Günther R. Raidl · Bin Hu · Christian Kloimüllner**

the date of receipt and acceptance should be inserted later

**Abstract** We consider a transportation problem arising in public bicycle sharing systems: To avoid rental stations to run entirely empty or full, a fleet of vehicles continuously performs tours moving bikes among stations. In the static problem variant considered in this paper, we are given initial and target fill levels for all stations, and the goal is primarily to find vehicle tours including corresponding loading instructions in order to minimize the deviations from the target fill levels. As secondary objectives we are further interested in minimizing the tours' total duration and the overall number of loading actions. For this purpose we first propose a fast greedy construction heuristic and extend it to a PILOT method that evaluates each candidate station considered for addition to the current partial tour in a refined way by looking forward via a recursive call. Next we describe a Variable Neighborhood Descent (VND) that exploits a set of specifically designed neighborhood structures in a deterministic way to locally improve the solutions. While the VND is processing the search space of candidate routes to determine the stops for vehicles at unbalanced rental stations, the number of bikes to be loaded or unloaded at each stop is derived by an efficient method. Four alternatives are considered for this embedded procedure based on a greedy heuristic, two variants of maximum flow calculations, and linear programming. Last but not least, we investigate a general Variable Neighborhood Search (VNS) and variants of a Greedy Randomized Adaptive Search Procedure (GRASP) for further diversification and extended runs. Rigorous experiments using benchmark instances derived from a real-world scenario in Vienna with up to 700 stations document the performance of the suggested approaches and individual pros and cons. While the VNS yields the best results on instances of moderate size, a PILOT/GRASP hybrid turns out to be superior on very large instances. If solutions are required in short time, the construction heuristic or PILOT method optionally followed by VND still yield reasonable results.

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

## 1 Introduction

In many cities around the world public Bicycle Sharing Systems (BSSs) have been introduced in the last decade. Such systems augment public transport very well and frequently present themselves as an attractive "green" alternative to individual motorized traffic. In addition, by providing an incentive for doing sports they are a significant contribution to improving public health [7].

Modern bicycle sharing systems consist of a collection of rental stations that are strategically distributed over the service area. At each station there is a self-service computer terminal and several bike parking positions. Registered users can easily rent a bike and return it at any other station and any time they want. This freedom of the users poses an important challenge for operators of BSSs. Over time, different factors cause an uneven distribution of bikes in the system because the numbers of bikes rented and returned, respectively, can differ significantly among the stations. Such conditions might be temporary, e.g., commuting patterns across a working day, or persistent, e.g., due to topographical factors [21]. The situation becomes critical when stations run completely empty or full and user demands cannot be fulfilled anymore. In the first case a prospective customer is turned away from using the system at all because no bikes are available to rent. In the second case, which might be even worse for the customer, he is forced to take a detour to find another station that still has free parking positions.

To avoid or at least reduce the probability for such unpleasant occurrences that greatly impact user satisfaction, the BSS operator needs to actively rebalance the system by redistributing bicycles between stations by a fleet of vehicles. Typically, operators use cars with trailers to pick up bicycles at stations that tend to become full and to move them to stations that run empty. In the *Balancing Bicycle Sharing System* (BBSS) problem we aim at finding efficient vehicle routes with corresponding loading instructions for bicycles at each visited station. The main goal is to put the system into a state that is as balanced as possible under consideration of a time limit for the rebalancing operation.

In this work we improve and extend our previous preliminary study [20] where we already addressed this problem by a Variable Neighborhood Search (VNS) with an embedded Variable Neighborhood Descent (VND) [15]. Now we additionally consider a new, more sophisticated construction heuristic based on the PILOT method [27] for obtaining initial solutions. For the VNS/VND, we present a more detailed analysis of four different strategies to derive loading instructions for routes. Furthermore, we randomize the construction heuristics and iteratively apply them in combination with the VND, yielding two variants of a Greedy Randomized Adaptive Search Procedure (GRASP) [22]. A new benchmark suit including very large instances with up to 700 stations is used for testing. These instances are derived from real data provided by Citybike Wien, the major public BSS in Vienna, Austria. It turns out that in comparison the VNS performs particularly well on medium-sized instances while the PILOT/GRASP combination yields the best results especially on larger instances. If only short run times are possible, the construction heuristics optionally followed by VND also provide very reasonable results.

The article is organized as follows: The next section formalizes the problem, while Section 3 reviews previous and related work. Section 4 describes the construction heuristic and its extension to a PILOT method. Four alternative methods for deriving loading instructions from candidate tours are discussed in Section 5. Sections 6, 7, and 8 describe the VND, GRASP, and VNS approaches including the used neighborhood structures, respectively. Information on the test instances and the results of diverse experiments are given in Section 9. Finally, Section 10 draws conclusions and sketches promising future work.

## 2 The Balancing Bicycle Sharing System Problem

We start by providing a formal definition of the BBSS problem. In this work we consider the static problem variant that neglects any user activities during the rebalancing process and where we strive to reach a target fill level of bikes that is pre-specified for each station. Suitable target fill levels are obtained in practice from a statistical demand forecast model that considers several aspects such as season, day, time, as well as the weather forecast [23]. This is another major research issue that exceeds the scope of the current article. By using such models operators are able to estimate reoccurring demands quite well in order to derive expected target values. Note that in most practical scenarios this static case of BBSS is already a useful approximation, since stations are usually designed sufficiently large in order to compensate short-term fluctuations. However, the balancing is still necessary because imbalances arise over longer time horizons, such as one or several days.

The BSS is represented by a complete directed graph $G_0 = (V_0, A_0)$. Node set $V_0 = V \cup \{0\}$ consists of nodes for the rental stations $V$ and a node 0 for the depot (i.e., parking place of the vehicles). Each arc $(u,v) \in A_0$ has associated a time $t_{u,v} > 0$. This value not only includes the time needed for traveling from $u$ to $v$, but also an expected average time needed for parking, handling the local computer terminal, and loading or unloading bikes at $v$. 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 three values: The capacity $C_v \geq 0$, i.e., the number of available bike parking positions, the number of available bikes at the beginning of the rebalancing process $p_v$, and the target number of bikes that should ideally be available after rebalancing $q_v$, with $0 \leq p_v, q_v \leq C_v$.

The BSS operator has a fleet of vehicles $L = \{1, \ldots, |L|\}$ that is available for moving bikes between stations. Each vehicle $l \in L$ has a capacity to transport $Z_l > 0$ bikes simultaneously, a total time budget $\hat{t}_l$ within which it has to finish a route, i.e., the worker's shift length. Each route has to start and end at the depot 0. We assume that all vehicles start and finish their routes empty. A practical rationale behind this is that frequently vehicles are publicly accessible at the depot and bikes cannot be locked at the vehicles' trailers.

Solutions to the BBSS problem consist of two parts. The first one 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 $\rho_l$ representing the number of stations traveled to. Note that stations may be visited multiple times by the same or different vehicles. For reasonable solutions these multiple visits are necessary as the station capacities $C_v$ are sometimes much larger than the vehicle capacities $Z_l$.

The second part of a solution consists of 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 to be picked up ($y_{l,v}^i > 0$) or delivered ($y_{l,v}^i < 0$) at station $v$ at the $i$-th stop of vehicle $l$. Of course loading actions may only take place at visited stations, i.e., $\forall v \neq r_l^i : \; y_{l,v}^i = 0$, and thus, for simplicity we also write $y_l^i$ for $y_{l,r_l^i}^i$, i.e., if no station index is explicitly specified we assume the station to be the visited one ($r_l^i$).

Note that an option would be to further limit the domains of these loading instructions by the station capacities, i.e., $y_{l,v}^i \in \{-\min(Z_l, C_v), \ldots, \min(Z_l, C_v)\}$. We, however, stay more general and potentially allow vehicles meeting at a station to exchange bikes directly. Imposing a limit based on station capacities would be too restrictive in this case.

Several conditions must hold for a solution to be feasible: The number of bikes available at each station $v \in V$ always needs to be within $\{0, \ldots, C_v\}$. For any vehicle $l \in L$ the number of simultaneously transported bikes may never exceed the capacity $Z_l$, and the total tour

length $t_l$

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

is restricted by the time budget $\hat{t}_l$, $\forall l \in L$.

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

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

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 activities including the overall time required for traveling all routes. Therefore, our objective function is given by

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

where $\omega^{\text{bal}}, \omega^{\text{load}}, \omega^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms. Following the advice from experts at Citybike Wien, we assume that any improvement in balance is always preferred over decreasing the number of loading actions or reducing the work time, and to ensure this preference we use appropriate scaling factors. In all our tests we use the setting $\omega^{\text{bal}} = 1$ and $\omega^{\text{load}} = \omega^{\text{work}} = 1/100\,000$.

### 2.1 Monotonicity for Fill Levels of Stations

A natural simplification for the BBSS problem is the restriction to *monotonicity* regarding the fill levels of stations. By exploiting it we will see that algorithms for deriving good or optimal loading instructions for given tours become simpler while in general solutions are not substantially worse in comparison to the general case.

Let $V_{\text{pic}} = \{v \in V \mid p_v > q_v\}$ denote *pickup stations*, i.e., the set of stations from which ultimately bikes should be removed, and $V_{\text{del}} = \{v \in V \mid p_v < q_v\}$ denote the set of *delivery stations*. The remaining stations $V \setminus V_{\text{pic}} \setminus V_{\text{del}}$ are initially already in balance.

In the monotonic case, vehicles are only allowed to load bicycles at pickup stations and unload them at delivery stations. In this way a station's fill level only decreases or increases monotonically, and consequently the order in which different vehicles visit a single station does not matter. Stations that are already balanced at the beginning do not need to be considered at all as no pickups or deliveries are allowed there.

While monotonicity appears to be a very intuitive simplification, enforcing it may exclude better solutions that, e.g., use stations as buffers to temporarily store bikes or by exchanging bikes between vehicles when they meet at some stations. An example of such a situation is shown in Figure 1.

Experiments in Section 9 will show that the impact of monotonicity on the objective values of solutions is recognizable but small. We assume that this trend also depends on the scaling factors in the objective function which put a substantially lower weight on the traveling time than on the imbalance. In practice, excellent solutions can be found even under the assumption of monotonicity.

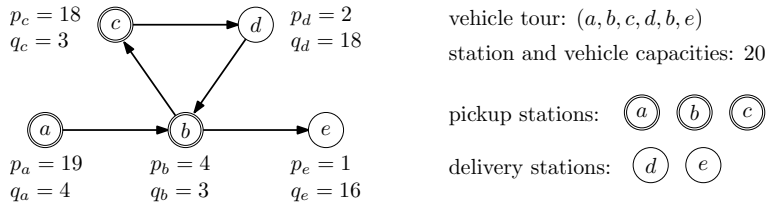



**Fig. 1** Example where the restriction to monotonicity yields a worse solution. With monotonicity, the best possible loading instructions are $y_1 = (+15, +1, +4, -16, 0, -4)$ resulting in a total imbalance of 22. In the general case, node $b$ can be used as buffer and loading instructions $y_1 = (+15,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

## 3 Related Work

Only in recent years the BBSS problem has been recognized as a combinatorial optimization problem and a few systematic solution approaches have been described by the operations research community. However, each concept addresses significantly different problem variants, making a direct comparison between existing approaches difficult. The majority of existing works use Mixed Integer Programming (MIP) techniques which in principle are sometimes able to find proven optimal solutions but in practice they are restricted to small instances regarding the number of stations and vehicles.

Chemla et al. [2] address the static case by using only one vehicle and exactly reaching the given target fill levels is defined as a hard constraint. No restriction is placed on the time needed for the rebalancing operation. The authors formulate an exact MIP model that appears to be intractable for realistic instances. Therefore, they modify the model by proposing a relaxation that is solved by a branch-and-cut approach, yielding a lower bound to the original problem. For obtaining a feasible solution and a corresponding upper bound, they employ a tabu search. To the best of our knowledge and with the exception of our previous work, this tabu search is the only metaheuristic approach applied to a variant of BBSS until now. A key concept in this work is the solution representation: Only the order in which the vehicle visits the various stations is considered. Loading instructions for each visit are obtained by an auxiliary algorithm based on a maximum flow computation, resulting in a greatly reduced search space. We adopt this principle but need to significantly extend it to suit our more general problem definition.

Raviv et al. [21] study two MIP-approaches for the static multiple-vehicle variant based on arc-indexed and time-indexed models, respectively. As objective function a non-traditional convex penalty function is used that is claimed to reflect user dissatisfaction by modeling a bounded birth and death process as well as calculating the expected number of shortage events in the system. This function is approximated in the MIP-formulations in a piecewise linear way. While the arc-indexed formulation is more compact it has some restrictions w.r.t. monotonicity. The time-indexed formulation is more flexible but requires a discretization of time and typically yields a substantially larger model. Tour lengths are also considered in the objective function, but the number of loading operations is ignored. Tests were performed on instances with up to 104 stations, one or two vehicles, and a time horizon of up to five hours. Solutions with practically reasonable optimality gaps could be obtained, especially with the arc-indexed approach.

Benchimol et al. [1] again assume balancing as hard constraint and only consider the total tour length as objective. They focus on approximation algorithms for selected special situations, for example by using the Christofides heuristic for the traveling salesman

problem. Their model comprises only a single vehicle and allows bikes to be temporarily dropped along the route before being moved to their final destination. Moreover, they give some complexity results for the cases when the graph representing the BSS is a tree or a line. The work focuses on theoretical aspects, so there is no experimental evaluation of the proposed approaches.

Contardo et al. [5] investigate the more complex dynamic scenario where rebalancing is done while activities in the bike sharing system cannot be neglected, e.g., during peak hours. 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 MIP-approach that utilizes Dantzig-Wolfe as well as Benders decomposition. This approach is able to handle randomly created instances with up to 100 stations and 60 time periods reasonably well. Upper and lower bounds can be derived relatively quickly, however significant gaps in the magnitude of 20–50% remain. Additionally applying branch-and-price only yields a small gain.

Chemla et al. [3] investigate concepts for the dynamic case of the BBSS problem. They describe a theoretical framework in order to study the dynamic problem and the vehicles' impacts on the system. Moreover, they prove that the dynamic BBSS problem is NP-hard and propose some heuristic approaches to solve the problem with a single vehicle. It is assumed that the city is already divided into sub-areas which are managed by only one vehicle, respectively. Finally, they describe a pricing technique, i.e., decreasing the bike's rent if the user returns the rented bike to a station which tends to run empty soon. Consequently, by applying this strategy it might be theoretically possible to omit vehicle tours completely.

Pfrommer et al. [17] also investigate the repositioning of bikes in an online scenario. They propose a heuristic for planning tours with multiple vehicles and test it in a simulation based on historic data. In addition, they also present a dynamic pricing strategy which encourages users to return bikes to empty stations. The truck tours and dynamic prices are periodically recomputed while the system is active.

In [20], we describe a greedy construction heuristic followed by a Variable Neighborhood Search/Variable Neighborhood Descent (VNS/VND) metaheuristic for efficiently finding vehicle routes. While the VNS searches the space of vehicle routes, corresponding loading instructions are efficiently derived by either a greedy method, maximum flow computations, or linear programming. We improved this work in [19], where a fourth alternative for deriving loading instructions is presented. Additionally, an effective combination of the four methods is considered. The current work extends our approaches by applying the PILOT method [27] in the construction heuristic and GRASP as an alternative to the VNS, as well as by performing extensive experiments on larger instances of up to 700 stations.

Di Gaspero et al. [9] describe a combination of Constraint Programming (CP) and Ant Colony Optimization to tackle the same BBSS variant as we do. Furthermore, in [8] the same authors suggest another CP approach utilizing a smart branching strategy and Large Neighborhood Search. In both works they tested with the benchmark suite we proposed in [20] and conclude that our VNS performs in almost all cases clearly better than their CP-based approaches. In particular, they show that it is quite difficult to deal effectively with the determination of loading instructions and the possibility of multiple visits in CP.

Finally, Schuijbroek et al. [25] decompose 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 CP model that represents the problem as a scheduling issue. Results on instances with up to 135 stations and five vehicles show that the approaches outperform a MIP model operating on the full unclustered problem. However, their model defines the target values as intervals

which must be fulfilled for every station. Consequently, they only minimize the total tour lengths of the vehicles.

There are further related works which focus on strategic planning aspects of BSSs such as location and network design depending on demands or on determining path distances between stations (e.g., [13,14]). Others study the system characteristics and usage patterns (e.g., [16]). However, these aspects are not within the scope of this work.

More generally, BBSS is related to diverse variants of the classical capacitated vehicle routing problem (VRP), see e.g., [10]. However, it differs in substantial ways: Most importantly, there are pickups and deliveries, and stations may be visited multiple times, even by different vehicles. Consequently, BBSS may be referred to as a capacitated single commodity split pickup and delivery VRP with multiple visits. BBSS is also related to the pickup and delivery traveling salesman problem (PDTSP) [11] where the goal is to find a cost-minimal route where goods are transported from pickup to delivery nodes. The PDTSP in turn is an extension of the selective traveling salesman problem or orienteering problem [6,12,26] where the objective is to find a route of limited length through a graph that maximizes the profits of the contained vertices.

Concerning the computational complexity of BBSS, it is trivial to show that the traveling salesman problem can be modeled as a special case of BBSS. Consequently, BBSS is also NP-hard in the strong sense and no polynomial-time constant-factor approximation can exist unless P=NP.

## 4 Construction Heuristics

We present two construction heuristics aimed at generating meaningful initial solutions within short time. The first basic heuristic, presented in the following subsection, has already been used in [20] and follows a classic greedy principle, but utilizes a greedy function specifically designed for BBSS. While fast, local greedy decisions can be far from optimal with regard to the whole solution. This is especially true for BBSS as the greedy function is a compromise that combines multiple objectives. To mitigate this problem, we extend the basic heuristic by evaluating each candidate station considered for addition to a partial tour in a deeper way by also considering its potential successors via recursive calls. This second approach follows the PILOT method [27] and is described in Section 4.2. Both methods assume monotonicity regarding fill levels of stations as defined in Section 2.1.

### 4.1 Greedy Construction Heuristic (GCH)

This greedy method builds solutions by iteratively creating a tour for each vehicle following a local best successor strategy. From the last station of a partial tour (or initially the depot), we first determine the set $F \subseteq V$ of feasible successor stations. Set $F$ includes all stations that are not yet balanced and additionally can be serviced by the current vehicle $l$ without exceeding the shift length $\hat{t}_l$, i.e., there is enough working time left to visit the station and to go back to the depot.

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

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

where $b_l$ represents the final load of vehicle $l$ so far and $a_v$ the final number of bikes at station $v$ in the currently considered partial tour. For an empty tour (i.e., $\rho_l = 0$) they are initialized with $b_l = 0$ and $a_v = p_v$, respectively. If routes for other vehicles have already been constructed, $a_v$ is modified to correctly reflect the number of available bikes under consideration of the other vehicles' actions.

We assume that no bikes are allowed to remain on a vehicle when returning to the depot. Therefore, an additional correction is important for pickup stations. For this purpose, we determine an estimation of the number of bicycles $b^{\text{del}}$ which can still be delivered to successive stations after visiting the last station within the remaining time. This is achieved by a recursive call of the construction heuristic which only considers delivery stations and assumes to have an unlimited amount of bicycles available at the vehicle.

Note that here we deviate in a detail from the greedy heuristic in [20]: In that work, the estimation of deliverable bicycles is individually determined for each candidate station $v \in F \cap V_{\text{pic}}$ considering it as the starting point. Tests indicated that the higher precision gained by these individual calculations is relatively small while the computational effort is substantially higher by a factor of $O(|V|)$. Especially when considering the extension to the PILOT method in the next chapter and the larger instances with up to 700 stations used here, the differences in running time become dramatical, and thus, we rely on the described simpler approach.

Having determined $b^{\text{del}}$, we discard all remaining pickup stations from $F$ if $a_v \geq b^{\text{del}}$, because in this case further pickups appear to be not possible anymore; i.e., the construction of the route is finished with delivery stations only. Otherwise, e.g., if further pickups are allowed, the number of bicycles to be collected at each candidate pickup station $v \in F \cap V_{\text{pic}}$ is corrected by considering the limit $b^{\text{del}}$:

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

Having calculated $\gamma_v$ for all candidate stations $v \in F$, we finally evaluate them by the ratio $\gamma_v / t_{u,v}$, where $t_{u,v}$ is the time needed to travel from the vehicle's last location $u$ to station $v$ and service $v$. Thus, this greedy evaluation criterion considers the balance increase per time unit. The node $v \in F$ with the highest ratio is then appended to the tour $r_l$; ties are broken randomly. Loading instructions are set as follows:

$$y_{l,v}^{\rho_l} = \begin{cases} \gamma_v & \text{if } v \in V_{\text{pic}} \text{ and} \\ -\gamma_v & \text{if } v \in V_{\text{del}}. \end{cases} \tag{6}$$

Furthermore, $b_l$ 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, i.e., $F = \emptyset$.

As $b^{\text{del}}$ is only an estimation, it may occasionally happen that a few bicycles remain in the vehicle at the end of a route. As we do not allow this in feasible solutions, we repair the situation by reducing the last pickup(s) correspondingly. If some $y_l^i$, $i = 1, \ldots, \rho_l$, becomes zero, then we remove visit $i$ from the route.

### 4.2 PILOT Construction Heuristic

The PILOT construction heuristic extends the greedy construction heuristic using the PILOT (Preferred Iterative LOok ahead Technique) method according to [27]. On several occasions,

this metaheuristic has already shown to yield better solutions than its simple greedy counterpart with only moderate and scalable computational overhead. In particular, we consider it to be a promising alternative to the VNS/VND approach for large instances where the VND might already take very long in execution. The basic idea of this method is to look ahead in order to escape the greedy trap, i.e., to further evaluate every candidate successor in a greedy way and thus avoid short-sighted results. The main issue of the greedy construction heuristic is that it always chooses the single locally best successor as long as the solution remains feasible. As a result, e.g., a dense cluster of stations which is in a greater distance from the current station than an isolated single station might yield a larger balance gain altogether, but the simple greedy algorithm does not recognize the cluster's overall benefit and selects the isolated station as successor. Contrarily, the PILOT construction heuristic evaluates each candidate station not just by its own distance and balance gain, but instead also in possible future gains by visiting further stations in corresponding recursive calls. To some degree, the PILOT approach is also related to probing techniques in Mixed Integer Programming [24].

Figure 2 shows the basic idea of PILOT in the context of BBSS. The vehicle is currently at station 1 and we evaluate all potential successors by greedily determining individual extensions with them. In this example we only show the evaluation for the stations $\{2,3,4\}$. It is performed by trying to temporarily append each candidate station to the current route and continuing the basic greedy construction process until no further station can be added. Furthermore, the constructed extensions are evaluated on a defined criterion which is in our case the total decrease of the objective function value (3). Finally, the candidate station with the highest benefit (i.e., objective function decrease) is selected – in our example station 3 – and appended to the route; all temporary solutions are discarded and PILOT continues with the next round of successor evaluations until $F$ becomes empty and the route is completed.

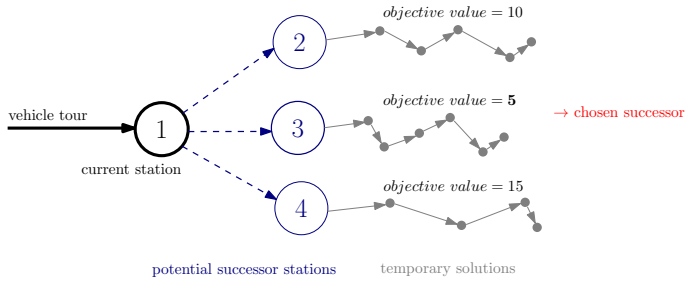
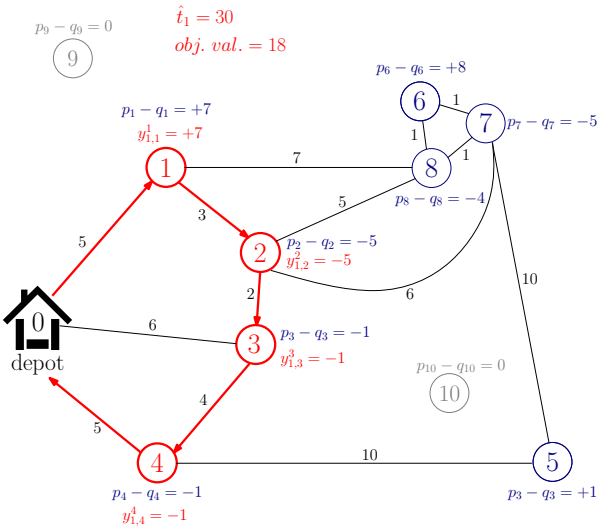


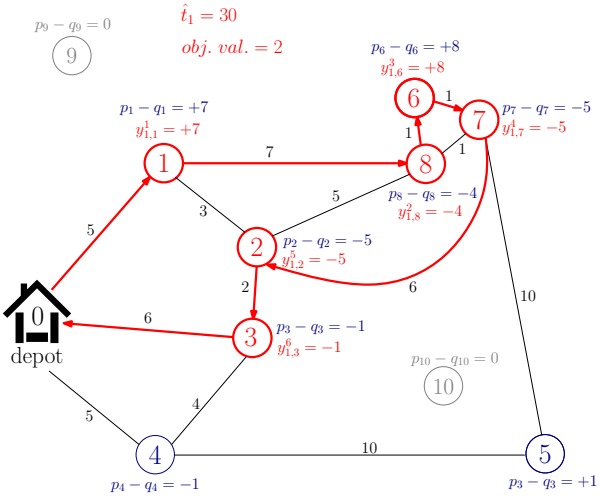

**Fig. 2** Basic principle of one iteration of the PILOT method for evaluating stations.

Note that the construction of the temporary extensions is done exactly the same way as in the basic greedy construction heuristic, including the calculation of the number of bicycles to be picked up or delivered, and taking into account the estimation of the number of bikes that can still be delivered.

Figure 3 shows an example how the PILOT approach dominates the simple greedy variant where the shift length is assumed to be $\hat{t}_1 = 30$min. For simplicity we only show the most lucrative connections and assume symmetric traveling times which are printed for each edge. The objective function values only show the imbalance and omit the other factors (working time and total number of loading instructions), in order to simplify the visualization. Figure 3a visualizes the solution of the greedy construction heuristic. Note that in particular the




(a) Greedy construction heuristic.

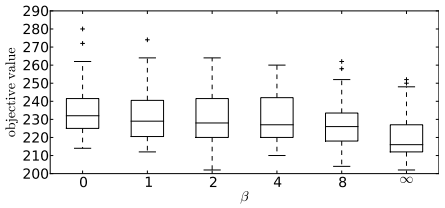



(b) PILOT method.

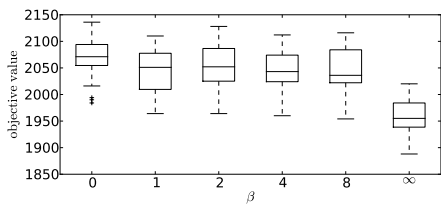
**Fig. 3** Exemplary solutions of the greedy construction heuristic and the PILOT method with one vehicle and and $\hat{t}_1 = 30$ min showing the benefits of the latter.

path from station 1 to 2 has a higher greedy value ($\frac{5}{3} = 1.67$) than to station 8 ($\frac{4}{7} = 0.57$), and again the path from station 2 to 3 is preferred over station 8. After the visit of station 4 no further feasible station is left. On the contrary, the PILOT method will select station 8 as second one because when considering it, the most lucrative extension with further stops at the stations 6, 7, 8, 2, and 3 is identified.

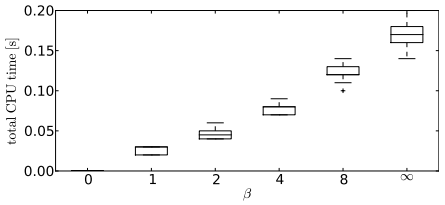
Due to the recursive evaluation of candidates the time complexity of the PILOT approach is higher than the time complexity of the basic greedy heuristic by a factor of $O(|V|)$. One possibility to improve the running time while still following the general idea is to apply



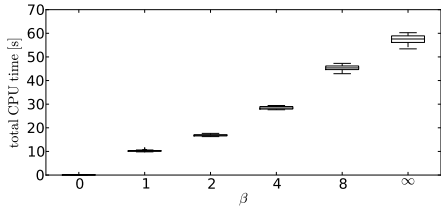

(a) Objective values for instances with $|V| = 90$, $|L| = 2$, and $\hat{t}_l = 8$ h, $\forall l \in L$.




(b) Objective values for instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8$ h, $\forall l \in L$.




(c) CPU times for instances with $|V| = 90$, $|L| = 2$, and $\hat{t}_l = 8$ h, $\forall l \in L$.




(d) CPU times for instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8$ h, $\forall l \in L$.

**Fig. 4** Finally best objective values and CPU times in seconds for different PILOT depths $\beta$.

a short-cut policy, i.e., to limit the recursive look-ahead to a certain number of successor stations, which is referred to as the *PILOT depth* $\beta$. In such a limited-depth PILOT approach, we do not evaluate each candidate extension by the overall gain in the objective function since the required time becomes a crucial factor again. Instead, we follow the criterion of the greedy heuristic, i.e., use the ratio of the balance gain and the time for the whole extension.

We tested our PILOT extension with various restricted depths and the unrestricted case on our benchmark instances, which are introduced in more detail in Section 9. Figure 4 shows the objective values and computation times for varying $\beta$ on benchmark instances including 700 and 90 stations, where $\beta = 0$ represents the simple greedy approach and $\beta = \infty$ the unrestricted depth. Since the unrestricted case still runs very fast compared to our other metaheuristics and yields significantly better results than when imposing any depth limit, we finally decided to only consider the unrestricted case in all further work.

## 5 Solution Representation and Deriving Loading Instructions

Our VND, GRASP, and VNS metaheuristics will be described in detail in Sections 6 to 8 and use an incomplete solution representation inspired by [2]. They process the search space of vehicle routes, while corresponding loading instructions $y^i_{l,v}$, $l \in L$, $v \in V$, $i = 1, \ldots, \rho_l$, are derived for each candidate solution by an embedded procedure. We consider four alternative methods for calculating loading instructions for a given set of routes $r$. The next sections describe them and examine their individual assets and drawbacks.

### 5.1 Greedy Heuristic (GH)

This fast heuristic approach follows the strategy from the greedy construction heuristic for a whole solution. It processes the routes vehicle by vehicle in a sequential way. Stations are

considered in the order as they are visited and loading instructions are computed in a similar way as described in Section 4.1. If the current station $v = r_l^i$, $l = 1,\ldots,|L|$, $i = 1,\ldots,\rho_l$, is a delivery station, then

$$y_{l,v}^i = -\min(q_v - a_v, b_l) \qquad (v \in V_{\text{del}}), \tag{7}$$

with $a_v$ indicating the current number of bikes at station $v$ and $b_l$ the number of currently loaded bikes at vehicle $l$. In case of $v$ being a pickup station, an estimation $b^{\text{del}}$ of the number of bikes which can still be delivered is calculated, but now on the basis of the already known successive delivery stations in the route. Loading instructions are then set to

$$y_{l,v}^i = \min(a_v - q_v, Z_l - b_l, b^{\text{del}} - b_l) \qquad (v \in V_{\text{pic}}). \tag{8}$$

GH is able to calculate loading instructions very quickly, but it is, as the construction heuristic, restricted to the monotonic case of the BBSS problem, i.e., does not make use of temporarily buffering bikes at stations or exchanging of bikes among vehicles. However, also under the assumption of monotonicity, GH is not guaranteed to find optimal loading instructions. For example, in a route where a station $v$ is visited twice, it can be beneficial to retain bikes in the vehicle at the first visit of $v$ in order to be able to satisfy a following delivery station. Station $v$ may later be also satisfied on its second visit.

### 5.2 Maximum Flow Approach for the Monotonic Case (MF-MC)

The MF-MC approach assumes monotonicity like GH, but it is an exact method, i.e., it always derives proven optimal loading instructions for a given set of routes. We apply a maximum flow computation on a specifically defined flow network. The approach is similar to [2], but we extend this method to our problem definition by considering multiple vehicles and handling balance as a goal in the objective function instead of a hard constraint. The design of the flow network implicitly enforces all constraints of the BBSS problem with regard to the number of bikes present in the stations and vehicles.

We define the 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 $\sigma$ and $\tau$ are the source and target nodes of the flow network, 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. The 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 $p_v - q_v$ representing the surplus number of bikes at each pickup station.
- $A_\tau = \{(v, \tau) \mid v \in V_{\text{del}}\}$ with capacities $q_v - p_v$ representing the lacking number of bikes at each delivery station.
- $A_{\text{pic}} = \{(v, v_l^i) \mid v_l^i \in V_L,\ v = r_l^i,\ v \in V_{\text{pic}}\}$, i.e., each pickup node in $V_{\text{pic}}$ 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 $V_{\text{del}}$. 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 given by the vehicle capacities $Z_l$.

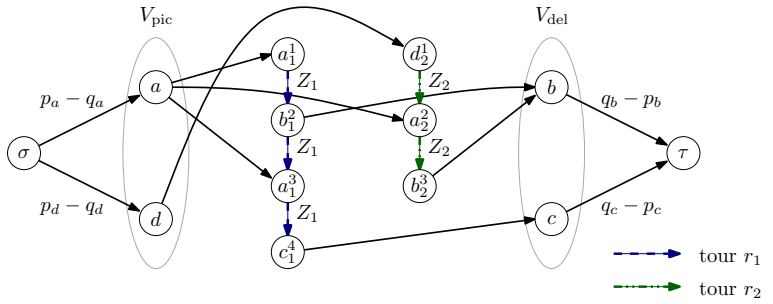



**Fig. 5** Exemplary flow network under the assumption of monotonicity for the tours $r_1 = (a,b,a,c)$ and $r_2 = (d,a,b)$ with $V_{\text{pic}} = \{a,d\}$ and $V_{\text{del}} = \{b,c\}$.

An exemplary network for an instance with four stations and two vehicles is shown in Figure 5. It can be seen easily that calculating a maximum $(\sigma,\tau)$-flow on the network directly yields (under the assumption of monotonicity) optimal loading instructions $y_l^i$ via the flows on the corresponding arcs $A_{\text{pic}}$ and $A_{\text{del}}$, respectively. In our implementation, we use the efficient push-relabel method from Cherkassky and Goldberg [4] for the maximum flow computations.

### 5.3 Linear Programming Approach (LP)

In the more powerful LP approach we are able to determine optimal loading instructions for the general, not necessarily monotonic case by solving a minimum cost flow problem on another network by linear programming (e.g., the network simplex algorithm). The main difference is that we now consider the order in which vehicles make their stops (at possibly the same stations). In this model, 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 $\sigma$ and $\tau$ we have a node $v^j$ for each station $v$ and time $j$ when a vehicle arrives at $v$. Furthermore, let $V^{\text{first}} = \{v^{j_{\min}} \in V_t \mid j_{\min} = \min\{j \mid v^j \in V_t\}\}$ denote the set of nodes representing the first visit of all stations among all routes and $V^{\text{last}} = \{v^{j_{\max}} \in V_t \mid j_{\max} = \max\{j \mid v^j \in V_t\}\}$, denote the set of nodes representing the last visit 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 time.
- $A_V = \bigcup_{v \in V} A_v$ 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 such a network is given in Figure 6. Now, a simple maximum $(\sigma,\tau)$-flow calculation would in general not yield optimal or even feasible loading instructions anymore as it must be guaranteed that all arcs $A_\sigma$ are satisfied (corresponding to the initially available bikes) and we do not have a correspondence between the achieved balance and

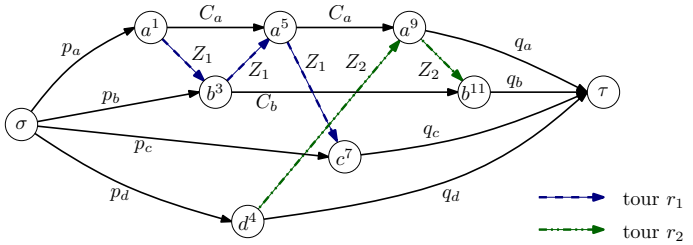



**Fig. 6** Exemplary flow network for the general case with tours $r_1 = (a,b,a,c)$ and $r_2 = (d,a,b)$.

the total flow. Instead, we have to minimize a certain objective function that depends on the flow, i.e., we have to solve the following minimum cost flow problem which is done by linear programming. Let the flow variables be $f_{u,v},\ \forall (u,v) \in A_\mathrm{f}$. By $pred_l(v^j) \in V_t$ we denote the predecessor of 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 $succ_l(v^j) \in V_t$ we denote its successor, i.e., $succ_l(v^j) = w^{j''}$ with $w = v_l^{i+1},\ j'' = t(r_l^{i+1})$. To calculate the balance as final absolute deviations of the target values and the total amount of loading operations, we split the variables for the loading instructions $y_{l,v}^i \in \{-Z_l,\dots,Z_l\}$ into $y_{l,v}^{+,i} \in \{0,\dots,Z_l\}$ for pickups and $y_{l,v}^{-,i} \in \{0,\dots,Z_l\}$ for deliveries of bikes, i.e., $y_{l,v}^i = y_{l,v}^{+,i} - y_{l,v}^{-,i},\quad y_{l,v}^{+,i} = 0 \ \vee\ y_{l,v}^{-,i} = 0$, and $|y_{l,v}^i| = y_{l,v}^{+,i} + y_{l,v}^{-,i}$.

$$\min \quad \omega^\mathrm{bal} \sum_{\forall v \in V^\mathrm{last}} \delta_v + \omega^\mathrm{load} \sum_{l \in L} \sum_{i=1}^{\rho_l} \left( y_{l,r_l^i}^{+,i} + y_{l,r_l^i}^{-,i} \right) \tag{9}$$

subject to

$$\sum_{(u,v^j) \in A_\sigma \cup A_V} f_{u,v^j} + \sum_{l \in L} \sum_{(u,v^j) \in A_{R,l}} f_{u,v^j} = \sum_{(v^j,w) \in A_\tau \cup A_V} f_{v^j,w} + \sum_{l \in L} \sum_{(v^j,w) \in A_{R,l}} f_{v^j,w} \qquad \forall v^j \in V_\mathrm{t} \tag{10}$$

$$y_{l,v}^{+,i} - y_{l,v}^{-,i} = \begin{cases} f_{v^j,succ_l(v^j)} & \forall l \in L,\ i = 1,\ v = r_l^i,\ j = t(r_l^i) \\ f_{v^j,succ_l(v^j)} - f_{pred_l(v^j),v^j} & \forall l \in L, i = 2,\dots,\rho_l - 1, v = r_l^i, j = t(r_l^i) \\ -f_{pred_l(v^j),v^j} & \forall l \in L,\ i = \rho_l,\ v = r_l^i,\ j = t(r_l^i) \end{cases} \tag{11}$$

$$f_{\sigma,v^j} = p_v \qquad \forall (\sigma,v^j) \in A_\sigma \tag{12}$$

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

$$q_v - f_{v^j,\tau} \le \delta_v \qquad \forall (v^j,\tau) \in A_\tau \tag{14}$$

$$0 \le f_{v^j,\tau} \le C_v \qquad \forall (v^j,\tau) \in A_\tau \tag{15}$$

$$0 \le f_{u^{j'},v^j} \le Z_l \qquad \forall l \in L,\ (u^{j'},v^j) \in A_{R,l} \tag{16}$$

$$0 \le f_{v^{j'},v^j} \le C_v \qquad \forall (v^{j'},v^j) \in A_V \tag{17}$$

$$\delta_v \ge 0 \qquad \forall (v^j,\tau) \in A_\tau \tag{18}$$

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

$$y^{-,i}_{l,v} \in \{0,\dots,Z_l\} \qquad \forall l \in L,\ v \in V,\ i = 1,\dots,\rho_l \quad (20)$$

The objective function (9) is directly derived from our BBSS objective (3). Equations (10) are the flow conservation equalities, while equations (11) 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 (12).

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.

### 5.4 Maximum Flow Approach for the General Case (MF-GC)

Since solving the above minimum cost flow problem on $G_f$ by linear programming is computationally expensive, we developed an alternative approach for obtaining the same optimal loading instructions based on two maximum flow calculations and an additional post-processing step; details of this rather complex procedure can be found in [19].

Although this approach, which we call here MF-GC, is computationally significantly more efficient than LP, it is still slower than MF-MC and especially GH. In preliminary results we observed that similar to the LP approach, the additional computational effort for allowing the solution to overcome the monotonicity restriction does not pay off in most cases. In this article we omit a detailed description but will include comparative results in Section 9.

In [19], we further evaluated a hybrid approach, in which the different strategies for calculating loading instructions are applied in a combined, adaptive way. In the VND, an additional neighborhood structure is used to determine the best suited method for a solution, and this method is inherited by its descendants. Results on instances with up to 90 nodes indicated small advantages for this approach. However, the benefits diminish for larger instances as considered in the current work, and running times become again considerably larger. Thus, we do not further consider the combination in this article.

## 6 Variable Neighborhood Descent (VND)

For locally improving candidate solutions, we employ several classical neighborhood structures that were successfully applied in various VRPs together with new structures exploiting specifics of BBSS within a Variable Neighborhood Descent [15]. All these neighborhood structures augment each other. Concerning the classical neighborhood structures, we based our design on the experience from [18].

The following neighborhoods are traversed in a best improvement fashion and applied in the given static order that has been determined experimentally. We also tried to use a dynamic reordering strategy but it did not yield any significant advantages.

After each move inside a neighborhood, all candidate tours that have changed are efficiently checked for feasibility with respect to time budgets using incremental computations. If one station appears multiple times in direct succession within a route, only the first stop is retained. Infeasible solutions, i.e., solutions where at least one vehicle route became infeasible, are discarded immediately. For solutions where all routes stay feasible we derive

loading instructions by one of the methods from the last section. Obsolete stops without any loading or unloading operations, i.e., where $y_l^i = 0$, are immediately removed from the routes.

**Remove station (REM-VND):** This neighborhood considers all possible removals of a single station in each route. Thus, a successful move avoids an unnecessary visit of a station: In this case, the same overall balance can be obtained without the visit, resulting in a shorter total working time and a higher potential to include some other station.

**Insert unbalanced station (INS-U):** This neighborhood tries to improve balance by considering each single yet unbalanced station for insertion at any position of any route.

**Intra-route 2-opt (2-OPT):** This is the classical 2-opt neighborhood from the traveling salesman problem applied individually to each route. Each possible segment of at least two stations is tried for inversion.

**Replace station (REPL):** Similarly to INS-U, this neighborhood considers stations which are currently unbalanced. However, it considers the replacement of an existing station by another unbalanced station.

**Intra or-opt (OR-OPT):** Here we consider solutions where sequences of one, two, or three stations are moved to another position within the same route.

**2-opt* inter-route exchange (2-OPT*):** This classical neighborhood of vehicle routing problems considers pairs of routes. All feasible exchanges of arbitrarily long end segments of the routes are enumerated. The neighborhood is implemented efficiently such that if an exchange of an end segment already resulted in an infeasible route, no end segments of larger length will be considered for moving to the route which became infeasible.

**Intra-route 3-opt (3-OPT):** This neighborhood structure resembles a restricted variation 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.

## 7 Greedy Randomized Adaptive Search Procedure (GRASP)

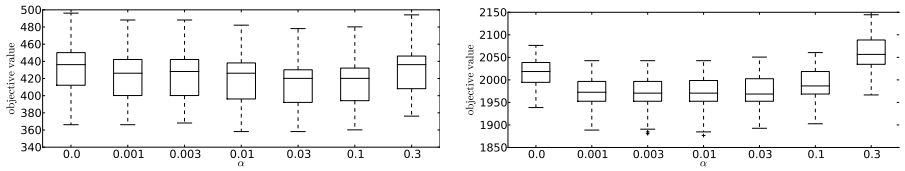
In order to prolong the heuristic search and obtain potentially better solutions, we extend our two construction heuristics to *Greedy Randomized Adaptive Search Procedures* (GRASP) according to [22]. For this purpose we iteratively apply a randomized version of the greedy or PILOT construction heuristic, respectively, and locally improve each solution with the VND. The overall best solution is returned.

The construction heuristics are randomized in order to obtain a diversified set of starting solutions for the VND. This randomization takes place in a GRASP-typical way: At each iteration of the construction heuristic, we do not always pick the locally best successor station but rank all candidates from $F$ (the serviceable, not yet balanced stations) according to the heuristic evaluation criterion. A restricted candidate list $RCL \subseteq F$ of best successors is preselected, and from these, one station is chosen uniformly at random. This successor is appended to the route, and the construction heuristic continues with its next iteration.

More precisely, the restricted candidate list $RCL$ contains the following elements:

$$RCL = \{v \in F \mid g(v) \geq g_{\max} - \alpha\,(g_{\max} - g_{\min})\}, \tag{21}$$

where $g(v)$ is the greedy value of candidate station $v$, while $g_{\max} = \max\{g(v) \mid v \in F\}$ and $g_{\min} = \min\{g(v) \mid v \in F\}$ are the maximum and minimum evaluation values that occur in




(a) Instances with $|V| = 180$, $|L| = 4$.

(b) Instances with $|V| = 700$, $|L| = 14$.

**Fig. 7** GCH-GRASP: Final objective values in dependence of $\alpha$; $\hat{t}_l = 8\text{h},\ \forall l \in L$.

$F$, respectively. Parameter $\alpha \in [0,1]$ controls the strength of the randomization, with $\alpha = 0$ resulting in a purely greedy solution, while $\alpha = 1$ turns the heuristic into a completely random construction method.

In preliminary tests on the benchmarks instances described in Section 9, we evaluated different values for $\alpha$ in a fixed and a randomized version. In the fixed version $\alpha$ remains constant throughout all GRASP iterations whereas in the randomized variant we choose an individual $\alpha'$ randomly from $[0,\alpha]$ for each GRASP-iteration. The randomized version turned out to be significantly more robust, and consequently we employ it in all further tests.

Moreover, our tests indicated that large instances w.r.t. $|V|$ tend to require smaller values for $\alpha$ than small instances. Figure 7 shows the impact of different values for $\alpha$ exemplarily for GCH-based GRASP on instances with $|V| = 180$, $|L| = 4$ and instances with $|V| = 700$, $|L| = 14$, and $\hat{t}_l = 8\text{h},\ \forall l \in L$. One can see that differences are generally relatively small, indicating the robustness of the method w.r.t. the choice of $\alpha$. Based on those preliminary tests we finally concluded to set

$$\alpha = 0.1 \cdot e^{-\frac{|V|}{200}} \tag{22}$$

in all following tests.

## 8 General Variable Neighborhood Search (VNS)

As an alternative to GRASP for diversification of the search, we embed the previously described VND into a Variable Neighborhood Search (VNS) as described in [15]. Similarly to the VND, the VNS neighborhood structures represent a combination of both classical neighborhoods from vehicle routing problems and more problem-specific neighborhoods of the BBSS problem.

In contrast to the VND, the VNS neighborhoods are generally larger and thus, they are not systematically searched, but instead used for *shaking*, i.e., for randomly deriving single new solutions in some distance to the incumbent solution. These solutions are always locally improved by the VND before the VNS decides upon their acceptance.

We use four types of VNS neighborhood structures, and each is parameterized by six different possible values of a parameter $\delta$, yielding a total of 24 individual neighborhoods. If a VNS move results in an infeasible solution containing routes that violate the available time budget of a vehicle, the respective routes are repaired by removing stations from the end. If the VNS does not find a better solution with the last neighborhood, it restarts with the first, until a termination criterion (i.e., CPU time or a certain number of iterations without improvement) is fulfilled.

**Move sequence (MV-SEQ$_\delta$):** This neighborhood selects a sequence of one to $\min(\delta, \rho_l)$ consecutive stations of a source route $r_l,\ l \in L$ randomly, and moves it to a random

position of a different route. If the original route contains less than $\delta$ stations, the whole route is inserted into the target route and the first route becomes empty. Both source and target routes are selected randomly, with the restriction that they must not be the same route. $\delta \in \{1,\ldots,5,\rho_l\}$.

**Exchange sequence (EX-SEQ$_\delta$):** This neighborhood also selects two routes at random. In each route, a randomly selected segment of length one to $\min(\delta,\rho_l)$ is chosen and exchanged with the respective other route. With a probability of 10% each exchanged segment is added to the target route in reversed order. This particular feature is adopted from [18]. $\delta \in \{1,\ldots,5,\rho_l\}$.

**Remove stations (REM-VNS$_\delta$):** Here we sequentially process all stops of all routes and remove each station visit with probability $\delta \in \{10\%,14\%,18\%,22\%,26\%,30\%\}$. We trust on the VND to again add fruitful visits.

**Destroy and recreate (D&R$_\delta$):** In this neighborhood we select a random position in a randomly chosen route $r_l,\ l \in L$. Then, all nodes from this position to the end of the route are removed and a new end segment is created by applying a randomized version of the PILOT construction heuristic. The randomization is done as described in Section 7, but with the threshold parameter set to $\alpha = \delta \in \{0\%,4\%,8\%,12\%,16\%,20\%\}$.

## 9 Computational Results

We performed extensive tests in order to assess the performance of our algorithms. After describing the way we generated our test instances, Section 9.2 compares the performance of VNS variants with different methods for deriving loading instructions and analyzes the efficiency of the VND neighborhood structures. We also use results from a sequence-indexed mixed integer programming (MIP) model as a baseline. For these preliminary analyses we only use small to medium-sized instances since a clear trend is visible and the MIP approach also reaches its limits. Finally, Section 9.3 compares the two construction heuristics with or without a subsequent VND run, as well as the GRASP and VNS approaches on all instances.

### 9.1 Benchmark Instances

We tested our algorithms on a new benchmark suite based on real-world data from the year 2011 provided by Citybike Wien[1] which runs a bike-sharing system with, at the time, 92 stations in Vienna, Austria. In order to evaluate the performance of the approaches on very large instances, our project partner Austrian Institute of Technology (AIT)[2] provided a larger set of 664 additional stations placed in Vienna at realistic positions. We generated the instances, which are available on the web[3], from the pool of 92 real plus 664 artificial stations as follows:

- Travel times $t_{u,v},\ (u,v) \in A_0$, are real average driving times from $u$ to $v$ plus an estimation for parking the vehicle and loading or unloading bikes at $v$.
- Station capacities $C_v$ of artificial stations were chosen randomly according to the distribution of real stations' capacities.

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

[2] http://www.ait.ac.at/

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

- The number of initially available bikes $p_v$ was taken from a snapshot of the system for all real stations. For the artificial ones, we first dedicated some of them as *support points* to which we assigned fill levels at random according to a certain distribution derived from the real stations. The fill levels of the remaining artificial stations were then determined by an Akima bicubic spline interpolation. In this way we achieve a small correlation between the fill levels of geographically close stations, as it can also be observed in the real data.
- Target values $q_v$ were derived from accumulated demands of the stations over a whole day which are known for the real stations. For terminals with a high number of bike demands we set the target value to 75% of the stations capacity, for a high number of parking position demands we set the target value to 25% and, if both are similar, then we set the target value to 50% of the capacity. For the artificial stations accumulated demands were determined randomly in a similar way as initial fill levels, i.e., to achieve a similar distribution and geographic correlation as in the real data.
- We derived instances with different numbers of stations $|V|$ by choosing the first station randomly from the pool of 756 stations and adding its $|V|$ nearest neighbors with regard to Euclidean distances. From the now $|V|+1$ stations, one was randomly selected to be the depot.
- In order to make complete balance at least theoretically possible when having enough working time and vehicles available, $\sum_{v\in V} p_v = \sum_{v\in V} q_v$ must hold in each instance. This was achieved by randomly selecting a station $v$ and incrementing or decrementing $p_v$ by one, as required. We iterated this process until the above equation was fulfilled. Note that there might not necessarily be the right number of bikes available to meet all target levels in reality. Still, operators strive to fulfill target levels by controlling the number of bikes in the system accordingly.
- We assume a homogeneous fleet of vehicles with capacities $Z_l = 20,\ \forall l \in L$.
- The time budget was set to $\hat{t} \in \{2,4,8\}$ hours, $\hat{t}_l = \hat{t},\ \forall l \in L$.
- The number of available vehicles $|L|$ varies and is stated in the following sections.
- For each configuration of $|V|$, $|L|$ and $\hat{t}$, 30 independent instances were created.

For the real-world scenario at Citybike Wien the configuration with $|L| = 2$ and $\hat{t} = 8$ is the most relevant. The planning for Citybike Wien's two vehicles is done in the morning and they are usually in action for a whole working day shift of eight hours. Considering shorter shift times thus becomes interesting when performing an "on-line" re-optimization for the remaining day after some major disruption only. Note that the considered static case is not exactly met here, as Citybike Wien operates 24/7. Nevertheless solving the static BBSS is still considered to be a reasonably good approximation as expected demands are relatively well known and the target values are set correspondingly. Furthermore, stations are designed with a significant reserve so that the balancing is not usually needed for solving short-term fluctuations but to resolve imbalances occurring over a longer time horizon, e.g., one or even more days. Citybike Wien has its highest activity in the evening and during the night.

Note that this benchmark suite is different from the one used in our former work [20]. While in the latter instances initial fill levels of artificial stations have been chosen simply at random and all target values have been set to 50% of the stations' capacities, the new instances are more realistic.

We implemented all algorithms in C++ using GCC 4.6. 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. For solving the LP-based approach to determine loading instructions CPLEX 12.4 was used with default settings, except for restricting CPLEX to only use a single thread.

As already mentioned in Section 2, the scaling factors in the objective function were set to $\omega^{\text{bal}} = 1$, $\omega^{\text{load}} = \omega^{\text{work}} = 1/100\,000$. 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. This aspect is a result of our discussions with project partner Citybike Wien. Reducing the lengths of routes and/or the number of loading operations can lower operation costs and has a positive environmental impact. In addition, tours with obviously redundant station visits or loading actions would strongly reduce the practical acceptance by the drivers. However, the top priority is still to balance the system so that target values are reached as far as possible and user satisfaction is maximized. From the operator's point of view, workers are paid for the whole shift length anyway, and therefore a reduction in the tour lengths is just a secondary aspect.

Objective values of different solutions must be compared with care due to the small scaling factors $\omega^{\text{load}}$ and $\omega^{\text{work}}$ for the secondary terms in the objective function. If two solutions achieve the same balance but differ regarding the secondary terms, the difference between the objective values will be very small although possibly important. Therefore, we list in the result tables for each algorithm variant and each instance set the number of runs for which the variant yielded the best results (#best) besides average objective values. We consider #best to be a better indicator for analyzing performance differences than average objective value differences. Maximum #best values are printed bold for each instance set.

## 9.2 Comparison of VNS Variants

In this section we analyze the influence of the four alternative procedures for deriving loading instructions within the VNS in Section 9.2.2 and further compare them to a MIP approach. Moreover, we study the performance of the VND neighborhood structures.

The following instance settings are used in these tests:

- The number of stations is $|V| \in \{10, 20, 30, 60, 90\}$.
- The vehicle fleet size is $|L| \in \{1, 2, 3, 5\}$.
- Each instance set uses a unique combination of $|V|$, $|L|$, $\hat{t}$ and contains 30 instances, resulting in a total of 30 sets and 900 instances.
- For each instance five independent runs were performed.
- Each run was terminated when no improvement could be achieved within the last 5 000 VNS iterations or after a CPU time of one hour. In the first case we consider the heuristic search as converged, major further improvements in prolonged runs are unlikely.

Tables 1 and 2 show aggregated results for these 30 instance sets. They also cover both the situation where perfect balance cannot be achieved due to a small number of vehicles and/or insufficient time budgets in relation to the number of stations, and the situation where perfect balance is possible. Table 1 shows for all approaches the number of runs where the respective approach obtained the best objective values (#best). For the MIP model, we additionally give mean upper bounds $\overline{\text{ub}}$, mean lower bounds $\overline{\text{lb}}$ and their respective standard deviations $\text{ub}_{\text{sd}}$, $\text{lb}_{\text{sd}}$. For each variant of the VNS, in addition to #best, we also list mean objective values $\overline{\text{obj}}$ and their standard deviations sd. Table 2 gives the median total run times $\widetilde{\text{t}_{\text{tot}}}$ for all approaches. For the VNS variants we also list the median number of performed VNS iterations $\widetilde{\text{g}_{\text{tot}}}$.

### *9.2.1 MIP models*

In addition to the metaheuristic algorithms, we implemented a MIP model based on the sequence-indexed formulation from [21] but adapted to our problem formulation in a straight-forward way. Just like GH and MF-MC, this model is not able to consider dependencies among vehicles and is therefore restricted to monotonicity. CPLEX 12.4 was used for trying to solve the instances with this model. Again, we used default settings, except for the restriction to use only a single thread. A CPU time limit of one hour was imposed. Furthermore, we investigated a second MIP model based on a time-indexed formulation [21] for the general case. Unfortunately, experiments indicated that this approach led to worse results than the first model due to the higher complexity of the model caused by the discretization of station visit times. Thus, we omit the results of the time-indexed formulation here.

We can clearly observe that the pure MIP approach is only able to solve the smallest instances to optimality within the given time limit. For most realistically sized instances very large gaps remain between upper and lower bounds. The MIP approach scales very badly with increasing numbers of vehicles and especially with longer vehicle time budgets. This is reflected in the low #best values for all but the smallest instances. For the larger instances CPLEX often only found trivial solutions where all vehicles are staying at the depot, or even no solutions at all. Also, no useful lower bounds can be derived for these more complex instances.

For the few instance groups with $|V| = 10$ where the MIP approach yielded small gaps, we observe that the VNS variants are almost always able to find solutions with equal or better objective values. Especially on instance sets with more than 10 stations, the VNS dramatically outperforms the MIP approach by obtaining better solutions in substantially shorter run times.

### *9.2.2 Comparing different variants for deriving loading instructions*

Among our four VNS variants, the one applying GH is clearly the fastest. MF-MC required about 120% more run time on average for each call of the auxillary algorithm for deriving loading instructions. LP is very slow, resulting in run times that are about 250 times longer than those of GH. MF-GC improves on the performance of LP, but is still around eight times slower than GH.

The solution quality of the VNS approach strongly depends on the ability to perform a substantial number of iterations. Therefore, a computationally more expensive variant needs to achieve rather large improvements in order to compete with the faster variants. We will now analyze the solution quality of the different variants for deriving loading instructions.

*VNS with GH:* As the fastest variant for deriving loading instructions, GH has a very high number of #best values (2969) that is only slightly exceeded by MF-MC. GH's total of the objective values is even better by a small degree. Especially on instance sets with large numbers of stations and shift lengths, this variant is able to achieve the highest #best values among all VNS variants. On small instances, GH performs slightly worse than MF-MC, but in general both variants obtain very similar results with regard to solution quality. In addition, GH has the advantage of significantly shorter run times.

*VNS with MF-MC:* This method shows the highest sum of #best values of all approaches (2987), as well as the second lowest total objective values. A Wilcoxon signed-rank test comparing MF-MC with the strongest contender GH on each instance set shows significant advantages with error probabilities of less than 5% for 18 of the 60 classes. We observe a clear tendency that MF-MC performs better on smaller instances while GH performs better on larger instances. There is no statistically significant difference between the two methods when looking at the total results of all 60 instance sets.

*VNS with LP:* In general the LP approach is able to construct better solutions than GH and MF-MC since it is not restricted to monotonicity but computes the optimal loading instructions for any given vehicle routes. However, this difference is barely visible in the results. We conclude that for the static BBSS problem the assumption of monotonicity does not have a practically significant negative impact on the solution quality. On the contrary, the LP approach suffers from the longest run times in comparison to the other variants and only a substantially smaller number of iterations can be performed within the time limit. About 60% of all runs were terminated before the VNS converged reasonably. This usually leads to worse objective values for LP in comparison to the other variants. When compared to GH, LP performs significantly worse when considering all 60 instance sets in total with an error probability of less than 0.1%. This is also reflected in the low sum of #best values for LP (2161).

*VNS with MF-GC:* Similarly to LP, MF-GC is theoretically able to construct better solutions than GH and MF-MC since it is not restricted to monotonicity. However, even with the vastly improved performance of MF-GC over LP, this difference is barely visible in the results. Again, higher computational expense of MF-GC compared to GH and MF-MC impedes the ability to find good solutions for larger instances because only a substantially smaller number of iterations can be performed within the time limit. While the #best values of MF-GC are quite good for small to medium instances, the faster methods outperform it for many instances with 60 and 90 stations, especially when the number of vehicles and/or shift length is large. A Wilcoxon test confirms this by showing that GH is significantly better than MF-GC for nine of the large instance sets and also in total over all instance sets with error probabilities of less than 1%. The total sum of #best values (2625) is about 10% lower than those of GH and MF-MC.

We conclude that it is more important to perform a large number of VNS iterations than to employ a more sophisticated but slower method for deriving loading instructions. While MF-GC and LP are theoretically more powerful than GH and MF-MC, they only infrequently lead to slightly better solution qualities and therefore cannot justify the higher computational effort. GH is by far the fastest method for deriving loading instructions, and no other method offers consistent significant advantages with regard to solution quality. Thus, GH is the best method for deriving loading instructions in practice.

**Table 1** Qualitative results of the MIP approach and the VNS considering the four variants of deriving loading instructions. Each instance set contains 30 instances, five runs per instance were performed.

| Instance set | | | MIP | | | | | VNS with GH | | | VNS with MF-MC | | | VNS with LP | | | VNS with MF-GC | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ [h] | #best | $\overline{\text{ub}}$ | $\text{ub}_{\text{sd}}$ | $\overline{\text{lb}}$ | $\text{lb}_{\text{sd}}$ | #best | $\overline{\text{obj}}$ | sd | #best | $\overline{\text{obj}}$ | sd | #best | $\overline{\text{obj}}$ | sd | #best | $\overline{\text{obj}}$ | sd |
| 10 | 1 | 2 | **150** | 27.801434 | 14.211779 | 27.801434 | 14.211779 | 145 | 27.868101 | 14.302006 | 145 | 27.868101 | 14.302006 | 140 | 28.001431 | 14.321390 | 140 | 28.001431 | 14.321390 |
| 10 | 1 | 4 | 95 | 3.469482 | 4.690047 | 0.175360 | 0.513799 | 132 | 3.536151 | 4.711462 | **142** | 3.509487 | 4.691593 | 130 | 3.536152 | 4.711463 | 141 | 3.522819 | 4.701557 |
| 10 | 1 | 8 | 35 | 0.003216 | 0.000455 | 0.002599 | 0.000462 | 135 | 0.003195 | 0.000453 | 146 | 0.003194 | 0.000452 | **150** | 0.003193 | 0.000452 | **150** | 0.003193 | 0.000452 |
| 10 | 2 | 2 | **150** | 9.136065 | 8.894377 | 8.804364 | 9.173847 | 140 | 9.269374 | 8.815557 | 145 | 9.269374 | 8.815557 | 145 | 9.269374 | 8.815557 | 141 | 9.322702 | 8.789374 |
| 10 | 2 | 4 | 110 | 0.003405 | 0.000564 | 0.003262 | 0.000542 | 145 | 0.003386 | 0.000559 | 148 | 0.003385 | 0.000559 | 149 | 0.003385 | 0.000559 | **150** | 0.003385 | 0.000559 |
| 10 | 2 | 8 | 115 | 0.003236 | 0.000500 | 0.003151 | 0.000426 | 130 | 0.003195 | 0.000453 | 143 | 0.003193 | 0.000452 | **145** | 0.003193 | 0.000452 | **145** | 0.003193 | 0.000452 |
| 20 | 2 | 2 | 64 | 52.469726 | 15.575644 | 29.779824 | 17.605633 | 140 | 51.629729 | 15.678652 | 139 | 51.643060 | 15.664408 | **146** | 51.523067 | 15.732142 | 140 | 51.589728 | 15.669937 |
| 20 | 2 | 4 | 0 | 16.005675 | 9.195814 | 0.003873 | 0.000536 | 79 | 5.272503 | 5.147186 | **101** | 5.245838 | 5.111881 | 79 | 5.299168 | 5.145725 | 99 | 5.245838 | 5.111880 |
| 20 | 2 | 8 | 0 | 0.141535 | 0.720372 | 0.003499 | 0.000540 | 110 | 0.006378 | 0.000615 | 120 | 0.006376 | 0.000615 | 93 | 0.006380 | 0.000614 | **126** | 0.006376 | 0.000616 |
| 20 | 3 | 2 | 6 | 34.737654 | 12.331921 | 2.087799 | 4.842072 | 117 | 28.791113 | 12.655299 | 120 | 28.751115 | 12.622697 | **124** | 28.964448 | 13.143979 | 118 | 28.857786 | 12.835286 |
| 20 | 3 | 4 | 0 | 0.941061 | 2.301619 | 0.005357 | 0.001232 | 72 | 0.006701 | 0.000591 | **106** | 0.006696 | 0.000589 | 86 | 0.006700 | 0.000588 | 96 | 0.006697 | 0.000590 |
| 20 | 3 | 8 | 0 | 8.142002 | 31.790072 | 0.003441 | 0.000519 | 104 | 0.006379 | 0.000615 | 121 | 0.006377 | 0.000615 | 91 | 0.006381 | 0.000614 | **126** | 0.006376 | 0.000616 |
| 30 | 2 | 2 | 15 | 112.869799 | 21.740922 | 73.209360 | 21.776380 | 145 | 110.003155 | 22.309574 | 143 | 110.003157 | 22.309576 | **147** | 109.843162 | 22.000016 | 143 | 110.029821 | 22.279456 |
| 30 | 2 | 4 | 0 | 72.272515 | 24.260691 | 0.005688 | 0.000644 | 71 | 34.659678 | 10.415634 | 72 | 34.659674 | 10.551349 | 51 | 34.939667 | 10.467490 | **75** | 34.779667 | 10.574958 |
| 30 | 2 | 8 | 0 | 192.733679 | 38.103015 | 0.003746 | 0.002333 | 61 | 0.009488 | 0.000590 | **95** | 0.009483 | 0.000585 | 17 | 0.009505 | 0.000589 | 83 | 0.009486 | 0.000588 |
| 30 | 3 | 2 | 0 | 91.904456 | 16.821933 | 27.790735 | 15.884329 | 107 | 79.551281 | 18.292426 | **111** | 79.257939 | 17.935839 | 108 | 79.177952 | 17.822698 | 104 | 79.431270 | 18.026606 |
| 30 | 3 | 4 | 0 | 39.208437 | 19.988591 | 0.005442 | 0.000690 | 53 | 6.035519 | 3.506518 | **60** | 6.008853 | 3.589846 | 19 | 6.462163 | 3.657592 | 56 | 5.942195 | 3.525073 |
| 30 | 3 | 8 | 0 | 198.533333 | 26.842485 | 0.000000 | 0.000000 | 74 | 0.009490 | 0.000591 | **82** | 0.009487 | 0.000585 | 20 | 0.009514 | 0.000591 | 75 | 0.009488 | 0.000589 |
| 60 | 3 | 2 | 0 | 276.304578 | 30.436585 | 176.901880 | 32.916749 | 117 | 253.351513 | 29.879859 | 115 | 253.231519 | 30.422328 | 119 | 253.138190 | 30.559345 | **120** | 253.138184 | 30.205040 |
| 60 | 3 | 4 | 0 | 395.200000 | 41.452164 | 0.000000 | 0.000000 | 43 | 119.103159 | 17.025314 | 51 | 119.023163 | 16.828918 | 5 | 121.689781 | 16.831045 | **61** | 118.983163 | 16.889415 |
| 60 | 3 | 8 | 0 | — | — | — | — | **88** | 6.084625 | 2.916488 | 65 | 6.324594 | 2.882422 | 0 | 10.457516 | 4.111593 | 6 | 7.137850 | 2.980819 |
| 60 | 5 | 2 | 0 | 302.137768 | 60.288079 | 53.576799 | 33.369663 | 66 | 185.661202 | 25.467138 | **77** | 185.327872 | 25.074180 | 43 | 186.207850 | 25.117945 | 64 | 185.541202 | 25.174178 |
| 60 | 5 | 4 | 0 | 395.200000 | 41.452164 | 0.000000 | 0.000000 | 58 | 37.895268 | 8.183309 | **81** | 37.615265 | 7.922555 | 0 | 43.241742 | 8.411625 | 14 | 39.068543 | 8.039271 |
| 60 | 5 | 8 | 0 | — | — | — | — | **80** | 0.019482 | 0.000674 | 71 | 0.019477 | 0.000683 | 0 | 0.019829 | 0.000718 | 8 | 0.019543 | 0.000674 |
| 90 | 3 | 2 | 0 | 514.336170 | 69.373279 | 253.108777 | 171.334278 | 119 | 429.711772 | 45.932137 | **122** | 429.458444 | 45.947627 | 113 | 429.618440 | 46.091014 | 113 | 429.578445 | 45.746608 |
| 90 | 3 | 4 | 0 | 594.333333 | 48.336303 | 0.000000 | 0.000000 | **62** | 262.837077 | 36.385916 | 56 | 262.957075 | 36.569536 | 5 | 267.370337 | 36.036094 | 42 | 263.290408 | 36.582442 |
| 90 | 3 | 8 | 0 | — | — | — | — | **104** | 77.326039 | 16.216842 | 47 | 78.046021 | 16.003556 | 0 | 86.672450 | 15.847739 | 2 | 80.405974 | 16.121748 |
| 90 | 5 | 2 | 0 | 594.333333 | 48.336303 | 0.000000 | 0.000000 | **77** | 348.061580 | 46.531085 | 64 | 348.248250 | 46.695211 | 36 | 349.634889 | 46.266354 | **77** | 348.381573 | 46.632770 |
| 90 | 5 | 4 | 0 | — | — | — | — | **81** | 140.469627 | 24.803457 | 65 | 140.842943 | 25.123944 | 0 | 150.882756 | 25.744827 | 7 | 143.282910 | 25.304140 |
| 90 | 5 | 8 | 0 | — | — | — | — | **114** | 0.801714 | 1.265215 | 34 | 1.148350 | 1.597211 | 0 | 8.000637 | 3.952740 | 3 | 1.974977 | 2.185755 |
| | | Total | 740 | 3932.221892 | 587.145678 | 653.276390 | 321.636453 | 2969 | 2217.987874 | 370.446215 | 2987 | 2218.507762 | 370.667375 | 2161 | 2263.999252 | 374.793550 | 2625 | 2227.574223 | 371.702839 |

**Table 2** Results regarding computation times and iteration counts of the MIP approach and the VNS considering the four variants of deriving loading instructions.

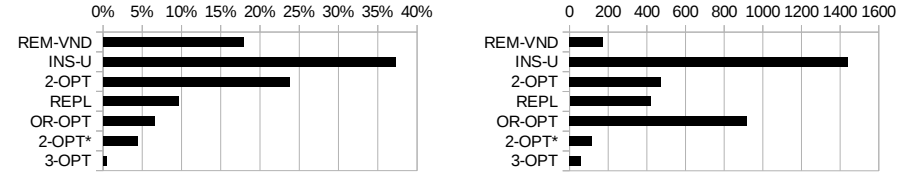
| Instance set | | | MIP | VNS with GH | | VNS with MF-MC | | VNS with LP | | VNS with MF-GC | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ [h] | $\widetilde{t_{tot}}$ [s] | $\widetilde{t_{tot}}$ [s] | $\widetilde{g_{tot}}$ | $\widetilde{t_{tot}}$ [s] | $\widetilde{g_{tot}}$ | $\widetilde{t_{tot}}$ [s] | $\widetilde{g_{tot}}$ | $\widetilde{t_{tot}}$ [s] | $\widetilde{g_{tot}}$ |
| 10 | 1 | 2 | 3.0 | 0.8 | 5001.0 | 1.8 | 5001.0 | 224.0 | 5001.0 | 5.3 | 5001.0 |
| 10 | 1 | 4 | 3600.0 | 4.8 | 5003.0 | 12.1 | 5003.0 | 1382.7 | 5004.5 | 41.3 | 5002.0 |
| 10 | 1 | 8 | 3600.0 | 7.8 | 5001.0 | 19.2 | 5001.0 | 2056.6 | 5001.0 | 70.5 | 5001.0 |
| 10 | 2 | 2 | 897.5 | 1.8 | 5004.0 | 4.1 | 5008.0 | 488.9 | 5007.0 | 14.0 | 5004.5 |
| 10 | 2 | 4 | 3600.0 | 5.8 | 5009.5 | 12.9 | 5005.0 | 1467.7 | 5006.0 | 47.4 | 5006.5 |
| 10 | 2 | 8 | 3600.0 | 6.7 | 5001.0 | 16.7 | 5001.0 | 1795.9 | 5001.0 | 60.7 | 5001.0 |
| 20 | 2 | 2 | 3600.0 | 4.3 | 5012.0 | 10.5 | 5014.5 | 1346.9 | 5021.5 | 38.7 | 5020.0 |
| 20 | 2 | 4 | 3600.0 | 38.9 | 5670.0 | 95.3 | 5456.0 | 3601.0 | 2320.5 | 358.1 | 5374.0 |
| 20 | 2 | 8 | 3600.0 | 75.4 | 5271.0 | 193.4 | 5177.0 | 3601.3 | 1197.5 | 773.2 | 5163.5 |
| 20 | 3 | 2 | 3600.0 | 8.4 | 5103.0 | 19.3 | 5151.5 | 2017.0 | 5109.5 | 79.8 | 5137.0 |
| 20 | 3 | 4 | 3600.0 | 41.8 | 5372.5 | 101.9 | 5510.5 | 3600.8 | 2273.0 | 385.5 | 5361.0 |
| 20 | 3 | 8 | 3600.0 | 72.4 | 5400.0 | 188.6 | 5282.0 | 3601.5 | 1304.5 | 680.2 | 5258.0 |
| 30 | 2 | 2 | 3600.0 | 6.9 | 5032.0 | 17.6 | 5019.0 | 1848.8 | 5020.0 | 61.5 | 5025.5 |
| 30 | 2 | 4 | 3600.0 | 63.4 | 5866.0 | 158.6 | 5657.5 | 3601.2 | 1503.0 | 635.2 | 5780.0 |
| 30 | 2 | 8 | 3600.0 | 242.9 | 5806.0 | 695.3 | 5932.0 | 3603.4 | 423.5 | 2664.1 | 5684.5 |
| 30 | 3 | 2 | 3600.0 | 12.8 | 5072.5 | 29.7 | 5087.5 | 3000.9 | 5035.0 | 117.5 | 5095.5 |
| 30 | 3 | 4 | 3600.0 | 130.4 | 6813.0 | 316.9 | 6612.5 | 3602.3 | 938.0 | 1346.8 | 7015.5 |
| 30 | 3 | 8 | 3600.0 | 238.4 | 6446.5 | 615.6 | 5978.0 | 3603.4 | 476.5 | 2523.3 | 6155.0 |
| 60 | 3 | 2 | 3600.0 | 29.7 | 5129.5 | 74.5 | 5124.5 | 3600.9 | 2612.0 | 289.5 | 5136.5 |
| 60 | 3 | 4 | 3600.0 | 301.1 | 7046.5 | 878.1 | 7705.0 | 3605.9 | 393.0 | 3474.4 | 6441.5 |
| 60 | 3 | 8 | 3600.0 | 3009.4 | 10407.5 | 3600.4 | 4269.5 | 3632.1 | 59.0 | 3602.1 | 1022.0 |
| 60 | 5 | 2 | 3600.0 | 75.6 | 6115.0 | 185.3 | 5726.0 | 3601.4 | 1441.0 | 760.1 | 5757.5 |
| 60 | 5 | 4 | 3600.0 | 1115.9 | 10492.0 | 2703.7 | 10387.5 | 3613.7 | 170.0 | 3600.7 | 3126.0 |
| 60 | 5 | 8 | 3600.0 | 2826.7 | 9488.0 | 3600.4 | 4325.5 | 3643.6 | 52.0 | 3601.8 | 979.5 |
| 90 | 3 | 2 | 3600.0 | 46.7 | 5200.0 | 136.3 | 5184.0 | 3601.0 | 1803.5 | 474.1 | 5185.5 |
| 90 | 3 | 4 | 3600.0 | 490.8 | 7343.0 | 1506.6 | 7326.0 | 3608.7 | 239.0 | 3600.5 | 4629.5 |
| 90 | 3 | 8 | 3600.0 | 3600.2 | 6914.5 | 3600.9 | 2401.5 | 3666.5 | 34.0 | 3604.0 | 552.0 |
| 90 | 5 | 2 | 3600.0 | 126.5 | 5924.5 | 330.2 | 5772.5 | 3602.3 | 809.5 | 1412.4 | 5923.5 |
| 90 | 5 | 4 | 3600.0 | 1856.5 | 10899.5 | 3600.2 | 7425.5 | 3626.1 | 97.5 | 3601.7 | 1705.5 |
| 90 | 5 | 8 | 3600.0 | 3600.7 | 3164.0 | 3602.0 | 1077.5 | 3721.7 | 15.0 | 3608.1 | 236.0 |
| | | Total | 101700.5 | 18043.5 | 185008.0 | 26328.1 | 162622.0 | 87968.2 | 68368.5 | 41532.5 | 136780.5 |

Up to this point we only used instances with up to 90 stations. However, it is already obvious that the more complex methods perform worse with increasing instance size. Since GH already turns out to be the best overall method on these instances, we refrained from testing even larger instances.

### *9.2.3 Performance of VND neighborhood structures*

Figure 8 shows relative success rates (i.e., the number of times a particular neighborhood structure was able to improve the solution divided by the number of applications) and CPU times of the VND neighborhoods for runs of the VNS with GH on the large instance set with $|V| = 90$, $|L| = 5$, $\hat{t} = 8$h. However, these results are typical for all instance sets except the smallest ones.

REM-VND is applied as the first neighborhood despite having lower relative success rates than the two following neighborhoods because it is meant to provide space in a route for the insertion of additional stations. Experiments confirmed that it is advantageous to use REM-VND as the first neighborhood directly followed by INS-U.




(a) Relative success rates.

(b) CPU times [s].

**Fig. 8** VND neighborhood performance for the instance set with $|V| = 90$, $|L| = 5$, $\hat{t} = 8$h using the VNS with GH.

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.

### 9.3 Comparing the GRASP/PILOT Hybrid with VNS

In this section we compare our GRASP/PILOT hybrid (see Section 7) with VNS with GH for deriving loading instructions. In order to analyze the different approaches in detail, we performed extensive tests across all instance sizes.

The following configurations were used in this section:

- The number of stations is $|V| \in \{30, 60, 90, 180, 300, 400, 500, 600, 700\}$.
- Meaningful numbers of vehicles $|L| \in \{1, \ldots, 21\}$ were chosen in dependence of $|V|$ and $\hat{t}_l$ and are listed in the result tables.
- Each instance set uses a unique combination of $|V|$, $|L|$, $\hat{t}$ and contains 30 instances, resulting in a total of 2310 instances grouped into 30 sets.

First of all, Table 3 compares the results of the basic greedy construction heuristic (GCH) and the PILOT method. It lists the number of runs where the respective approach obtained the best objective values (#best), the mean objective values $\overline{\text{obj}}$, and their median computation times $\widetilde{\text{t}_{\text{tot}}}$. According to a Wilcoxon signed-rank test with an error level of 5%, PILOT yields significantly better results on all aggregated results. This dominance can also be easily observed by looking at the objective values and #best. Although PILOT requires more computation time, even on the largest instances the additional effort does not get out of hand, and we consider it to be a good tradeoff. Consequently, for practical applications that require solutions in short time, the PILOT heuristic is a good choice.

In order to analyze the improvement potentials of the solutions obtained by the construction heuristics, we add a local improvement step via VND at the end and compare the results in Table 4. First we observe that PILOT with VND performs significantly better than GCH with VND, which is supported by the Wilcoxon signed-rank test with an error probability of less than 5%. The total run times of both approaches are very close now. The reason is that although GCH is much faster than PILOT, VND has to spend more iterations to reach a local optimum afterwards which evens up at the end. During these tests, PILOT is the clear winner.

Finally, Table 5 shows the final results for VNS, GCH-GRASP (GRASP with randomized greedy construction heuristic), and PILOT-GRASP (GRASP with randomized PILOT construction heuristic). Although we performed tests for the entire instances collection of 77 sets, the table only contains a selection of these sets. They represent the global trend and are sufficient for the conclusions.

**Table 3** Computational results of the greedy construction heuristic (GCH) and the PILOT method.

| Instance set | | | GCH | | | | PILOT | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ $[h]$ | #best | $\widetilde{\text{obj}}$ | sd | $\widetilde{\text{t}_{\text{tot}}}$ $[s]$ | #best | $\widetilde{\text{obj}}$ | sd | $\widetilde{\text{t}_{\text{tot}}}$ $[s]$ |
| 30 | 1 | 2 | 8 | 152.001530 | 26.203390 | $< 0.1$ | **30** | 147.934980 | 25.818900 | 0.0 |
| 30 | 1 | 4 | 1 | 105.336470 | 22.402790 | $< 0.1$ | **30** | 98.536640 | 20.824000 | 0.0 |
| 30 | 1 | 8 | 0 | 38.939550 | 11.694270 | $< 0.1$ | **30** | 33.206330 | 9.834630 | 0.0 |
| 60 | 1 | 4 | 2 | 283.403320 | 30.702870 | $< 0.1$ | **30** | 274.536880 | 30.974760 | 0.0 |
| 60 | 1 | 8 | 0 | 189.473390 | 23.429750 | $< 0.1$ | **30** | 181.006870 | 23.042860 | 0.1 |
| 60 | 2 | 2 | 3 | 309.069590 | 35.718110 | $< 0.1$ | **27** | 299.203200 | 33.768400 | 0.0 |
| 90 | 2 | 4 | 1 | 374.206650 | 39.689670 | $< 0.1$ | **29** | 358.273740 | 40.821710 | 0.1 |
| 90 | 2 | 8 | 0 | 204.479890 | 28.385090 | $< 0.1$ | **30** | 190.146840 | 28.724430 | 0.4 |
| 90 | 4 | 2 | 0 | 423.739160 | 54.371890 | $< 0.1$ | **30** | 397.606520 | 45.975640 | 0.0 |
| 180 | 4 | 4 | 0 | 772.613200 | 45.352750 | $< 0.1$ | **30** | 737.747390 | 45.958780 | 0.5 |
| 180 | 4 | 8 | 0 | 427.826300 | 34.797430 | $< 0.1$ | **30** | 395.827040 | 33.669300 | 2.9 |
| 180 | 5 | 8 | 0 | 317.565360 | 29.071670 | $< 0.1$ | **30** | 285.432860 | 28.725460 | 3.4 |
| 300 | 6 | 4 | 0 | 1321.753520 | 56.947210 | $< 0.1$ | **30** | 1271.554660 | 60.493890 | 2.0 |
| 300 | 6 | 8 | 0 | 782.906610 | 37.341860 | $< 0.1$ | **30** | 736.841050 | 35.493270 | 11.8 |
| 300 | 9 | 8 | 0 | 462.657110 | 25.704740 | $< 0.1$ | **30** | 413.858500 | 22.799850 | 15.4 |
| 400 | 8 | 4 | 0 | 1754.493320 | 75.600710 | 0.1 | **30** | 1681.628380 | 68.761200 | 4.4 |
| 400 | 8 | 8 | 0 | 1028.119890 | 47.776700 | 0.1 | **30** | 971.388000 | 41.910880 | 27.6 |
| 400 | 12 | 8 | 0 | 607.542700 | 35.657190 | 0.1 | **30** | 543.077910 | 32.626580 | 36.0 |
| 500 | 10 | 4 | 0 | 2205.566730 | 74.431490 | 0.1 | **30** | 2118.902180 | 70.022140 | 8.1 |
| 500 | 10 | 8 | 0 | 1301.733160 | 50.863390 | 0.1 | **30** | 1225.801700 | 44.761010 | 50.9 |
| 500 | 15 | 8 | 0 | 764.095080 | 34.038710 | 0.1 | **30** | 681.564200 | 29.478280 | 67.6 |
| 600 | 12 | 4 | 0 | 2712.906470 | 50.845290 | 0.1 | **30** | 2597.909120 | 40.370350 | 13.7 |
| 600 | 12 | 8 | 0 | 1608.079730 | 29.548010 | 0.2 | **30** | 1514.815390 | 27.303960 | 87.8 |
| 600 | 18 | 8 | 0 | 954.114180 | 31.063330 | 0.2 | **30** | 854.983800 | 26.979600 | 116.6 |
| 700 | 14 | 4 | 0 | 3115.779650 | 72.914380 | 0.2 | **30** | 2986.049480 | 62.149560 | 21.1 |
| 700 | 14 | 8 | 0 | 1866.959250 | 51.772820 | 0.2 | **30** | 1755.028720 | 41.214960 | 132.8 |
| 700 | 21 | 8 | 0 | 1104.599490 | 43.605020 | 0.2 | **30** | 984.469630 | 34.184230 | 178.5 |
| | | Total | 15 | 25189.961300 | — | 1.7 | 806 | 23737.332010 | — | 781.7 |

Due to the highly different characteristics of our metaheuristics and in order to compare them in a fair way, we use a common time limit ($\text{t}_{\text{max}}$) as the sole termination criterion. For the largest instances (180 to 700 stations) it is set to one hour, for medium instances (60 to 90 stations) we use 30 minutes, and for small instances (30 stations) we use 15 minutes. The median number of iterations $\widetilde{g_{tot}}$ suggests that all approaches scale well up to 700 instances.

For the GRASP variants we set $\alpha$ according to equation (22) in Section 7. We observe that PILOT-GRASP yields significantly better results than GCH-GRASP, which is supported by the Wilcoxon signed-rank test ($< 5\%$ error probability). We also see a clear improvement over the results of GCH-VND and PILOT-VND from the previous table.

As the main competitor for PILOT-GRASP, we chose VNS with GH since we concluded in Section 9.2.2 that this is the most robust VNS variant with the best balance between computation time and solution quality. Especially on larger instances which we are considering now, VNS with GH outperforms the other VNS variants. For the VND inside the VNS we use a fixed order of neighborhood structures as suggested in Section 6. However, for the GRASP runs the random VND order yields overall better results and we thus employ it for all GRASP tests. Comparing VNS with PILOT-GRASP, we observe an interesting trend. While all approaches perform comparably good on small instances with 30 nodes, VNS with GH dominates the medium-sized instances with 60 to 180 nodes. On large instances with 400 or more nodes, PILOT-GRASP is the clear winner.

In order to check if this trend depends on the chosen time limits ($\text{t}_{\text{max}}$), we also performed computational tests with shorter time limitations. Therefore, we apply the same gen-

**Table 4** Computational results for construction heuristics with local improvement via VND.

| **Instance set** | | | **GCH-VND** | | | | **PILOT-VND** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ $[h]$ | #best | $\overline{\text{obj}}$ | sd | $\widetilde{t_{tot}}$ $[s]$ | #best | $\overline{\text{obj}}$ | sd | $\widetilde{t_{tot}}$ $[s]$ |
| 30 | 1 | 2 | 22 | 148.601630 | 25.806420 | $< 0.1$ | **25** | 147.801650 | 25.764711 | 0.0 |
| 30 | 1 | 4 | 14 | 97.869980 | 20.537200 | $< 0.1$ | **24** | 97.736620 | 20.649725 | 0.0 |
| 30 | 1 | 8 | 9 | 32.339640 | 10.280110 | $< 0.1$ | **22** | 31.339650 | 9.661051 | 0.1 |
| 60 | 1 | 4 | 10 | 276.270150 | 29.556540 | $< 0.1$ | **22** | 273.670200 | 30.617129 | 0.0 |
| 60 | 1 | 8 | **17** | 178.406880 | 22.605600 | $< 0.1$ | 14 | 178.740170 | 22.806587 | 0.1 |
| 60 | 2 | 2 | 12 | 299.936470 | 32.798160 | $< 0.1$ | **24** | 297.736530 | 33.968484 | 0.0 |
| 90 | 2 | 4 | 6 | 361.007000 | 39.099410 | $< 0.1$ | **25** | 356.607050 | 40.592714 | 0.1 |
| 90 | 2 | 8 | 6 | 189.546730 | 29.401890 | 0.3 | **24** | 185.013490 | 28.761965 | 0.6 |
| 90 | 4 | 2 | 4 | 404.472980 | 46.165080 | $< 0.1$ | **26** | 396.339860 | 46.010674 | 0.0 |
| 180 | 4 | 4 | 2 | 749.213760 | 46.891200 | 0.2 | **28** | 735.547360 | 45.351396 | 0.6 |
| 180 | 4 | 8 | 5 | 400.026790 | 35.451870 | 1.9 | **25** | 392.560250 | 32.355673 | 3.8 |
| 180 | 5 | 8 | 5 | 289.299210 | 28.640580 | 3.8 | **25** | 281.632680 | 28.291244 | 5.4 |
| 300 | 6 | 4 | 3 | 1282.421030 | 59.696840 | 1.0 | **27** | 1265.487980 | 60.006868 | 2.4 |
| 300 | 6 | 8 | 1 | 745.240620 | 37.650540 | 6.4 | **29** | 729.374280 | 36.928317 | 15.4 |
| 300 | 9 | 8 | 2 | 424.057690 | 24.342590 | 19.4 | **28** | 407.191520 | 24.301169 | 25.4 |
| 400 | 8 | 4 | 0 | 1704.227880 | 71.566280 | 3.0 | **30** | 1674.895060 | 70.285444 | 5.6 |
| 400 | 8 | 8 | 1 | 980.987530 | 42.888140 | 16.2 | **29** | 965.054470 | 41.931984 | 34.9 |
| 400 | 12 | 8 | 0 | 557.743570 | 31.479010 | 54.5 | **30** | 536.144270 | 34.064915 | 58.7 |
| 500 | 10 | 4 | 0 | 2145.301500 | 73.154580 | 6.3 | **30** | 2110.102140 | 70.194881 | 10.5 |
| 500 | 10 | 8 | 0 | 1240.201050 | 48.395080 | 39.7 | **30** | 1216.801520 | 44.478994 | 65.5 |
| 500 | 15 | 8 | 0 | 699.629590 | 31.067280 | 107.8 | **30** | 673.563680 | 27.767719 | 117.0 |
| 600 | 12 | 4 | 0 | 2630.441670 | 45.924340 | 13.2 | **30** | 2590.242360 | 40.951831 | 17.7 |
| 600 | 12 | 8 | 0 | 1541.014560 | 27.596550 | 61.5 | **30** | 1506.548520 | 27.790606 | 109.0 |
| 600 | 18 | 8 | 1 | 880.182330 | 26.932450 | 183.0 | **29** | 847.450010 | 27.957043 | 172.1 |
| 700 | 14 | 4 | 1 | 3023.781870 | 66.177400 | 24.6 | **29** | 2977.782790 | 60.409126 | 26.2 |
| 700 | 14 | 8 | 0 | 1778.027740 | 45.659610 | 101.0 | **30** | 1743.695180 | 41.563750 | 167.0 |
| 700 | 21 | 8 | 0 | 1015.934540 | 40.265470 | 262.9 | **30** | 974.269160 | 34.392786 | 274.7 |
| | | Total | 121 | 24076.184390 | — | 906.7 | 725 | 23593.328450 | — | 1112.8 |

eral settings as before, however set the run time $t_{max}$ for instances with small instances (30 stations) to 5 minutes, for medium instances (60 to 90 stations) to 10 minutes and for largest instances (180 to 700 stations) to 15 minutes. Computational results of these short-time runs indicate as well that VNS dominates on medium instances and PILOT-GRASP on large instances. A Wilcoxon signed-rank test with an error level of 5% confirms this observation. Table 6 shows a few selected results from these tests. Naturally, the obtained solutions are usually worse than those of Table 5 with the larger computation time limits.

**Table 5** Computational results of VNS with GH and two GRASP variants with equal run times on a selection of the entire instances collection.

| Instance set | | | | VNS with GH | | | | GCH-GRASP | | | | PILOT-GRASP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ [h] | $t_{max}$ [s] | #best | $\overline{obj}$ | sd | $\widetilde{g_{tot}}$ | #best | $\overline{obj}$ | sd | $\widetilde{g_{tot}}$ | #best | $\overline{obj}$ | sd | $\widetilde{g_{tot}}$ |
| 30 | 1 | 2 | 900 | 28 | 147.201660 | 25.480765 | 1215760.5 | 19 | 148.268300 | 26.048913 | 2145205.5 | **29** | 147.135000 | 25.519076 | 618965.5 |
| 30 | 1 | 4 | 900 | **28** | 95.203360 | 19.886650 | 269610.5 | 15 | 96.336670 | 20.335746 | 164902.5 | 24 | 95.736670 | 20.582820 | 100840.0 |
| 30 | 1 | 8 | 900 | 27 | 29.206390 | 9.650602 | 37254.0 | 8 | 29.806360 | 9.845139 | 14294.5 | **28** | 29.273050 | 9.762614 | 10879.0 |
| 60 | 1 | 4 | 1800 | **30** | 269.603620 | 30.419674 | 310747.5 | 11 | 271.870230 | 30.148855 | 210631.5 | 18 | 271.203580 | 31.599269 | 77673.0 |
| 60 | 1 | 8 | 1800 | **26** | 170.473670 | 22.082650 | 44431.5 | 6 | 171.073660 | 22.082956 | 23064.5 | 16 | 171.006990 | 22.045548 | 10889.5 |
| 60 | 2 | 2 | 1800 | **25** | 293.803310 | 33.040634 | 505746.0 | 7 | 297.536530 | 31.825752 | 299233.0 | **25** | 294.069990 | 33.236838 | 190196.5 |
| 90 | 2 | 4 | 1800 | **26** | 346.273900 | 40.509658 | 50771.5 | 2 | 349.807160 | 40.166443 | 44325.0 | 7 | 348.740520 | 41.049992 | 17740.5 |
| 90 | 2 | 8 | 1800 | **23** | 173.147050 | 28.282862 | 7677.0 | 3 | 176.146990 | 28.420301 | 3566.5 | 4 | 175.347020 | 28.817433 | 2325.5 |
| 90 | 4 | 2 | 1800 | **23** | 386.740060 | 47.207416 | 131270.0 | 6 | 391.139920 | 48.325450 | 64951.0 | 16 | 387.540030 | 46.335124 | 43086.5 |
| 180 | 4 | 4 | 3600 | **28** | 718.080960 | 44.929107 | 14634.5 | 0 | 724.014190 | 44.884625 | 9261.5 | 3 | 722.614230 | 44.746593 | 5184.0 |
| 180 | 4 | 8 | 3600 | **25** | 374.227260 | 31.242738 | 1687.5 | 2 | 380.893790 | 33.121376 | 960.5 | 3 | 379.960480 | 32.680340 | 739.0 |
| 180 | 5 | 8 | 3600 | **28** | 264.033070 | 27.593286 | 996.5 | 0 | 272.832870 | 27.314928 | 558.5 | 2 | 269.832900 | 26.952757 | 510.0 |
| 300 | 6 | 4 | 3600 | **28** | 1241.288310 | 60.463810 | 3209.0 | 0 | 1255.754790 | 59.872953 | 1944.5 | 2 | 1248.821580 | 59.773513 | 1355.5 |
| 300 | 6 | 8 | 3600 | **15** | 715.107800 | 37.577745 | 322.5 | 4 | 719.241030 | 35.357035 | 254.0 | 11 | 715.641130 | 35.701784 | 201.0 |
| 300 | 9 | 8 | 3600 | 3 | 403.258240 | 25.367040 | 127.5 | 1 | 404.524820 | 23.528193 | 94.5 | **26** | 394.391770 | 22.652595 | 114.5 |
| 400 | 8 | 4 | 3600 | **24** | 1654.161990 | 68.573479 | 1179.5 | 0 | 1671.628390 | 69.262913 | 747.5 | 6 | 1658.495340 | 69.151828 | 612.0 |
| 400 | 8 | 8 | 3600 | 3 | 958.587940 | 40.943953 | 120.0 | 0 | 959.654480 | 41.520911 | 104.5 | **27** | 949.054670 | 41.763910 | 91.5 |
| 400 | 12 | 8 | 3600 | 0 | 543.477300 | 33.272886 | 53.0 | 1 | 540.343790 | 32.175396 | 37.0 | **29** | 524.344430 | 32.440027 | 52.0 |
| 500 | 10 | 4 | 3600 | 12 | 2092.169020 | 70.525137 | 517.0 | 0 | 2111.435410 | 69.433623 | 362.5 | **18** | 2091.902400 | 69.460983 | 322.0 |
| 500 | 10 | 8 | 3600 | 1 | 1225.468080 | 46.925477 | 55.5 | 2 | 1217.467930 | 45.867290 | 51.0 | **27** | 1202.935050 | 45.181073 | 49.5 |
| 500 | 15 | 8 | 3600 | 0 | 690.229840 | 34.079096 | 26.5 | 0 | 683.229880 | 29.705483 | 18.0 | **30** | 660.430570 | 29.605088 | 28.0 |
| 600 | 12 | 4 | 3600 | 4 | 2581.042510 | 37.595799 | 271.0 | 0 | 2596.042260 | 39.244476 | 176.0 | **26** | 2570.176080 | 39.503520 | 191.0 |
| 600 | 12 | 8 | 3600 | 0 | 1526.481530 | 27.629870 | 30.5 | 0 | 1515.214760 | 26.484485 | 30.0 | **30** | 1491.282070 | 28.096225 | 30.0 |
| 600 | 18 | 8 | 3600 | 0 | 872.849160 | 27.313824 | 23.0 | 0 | 869.915450 | 25.603318 | 10.5 | **30** | 835.650230 | 26.489396 | 18.0 |
| 700 | 14 | 4 | 3600 | 0 | 2985.649220 | 62.279120 | 157.5 | 0 | 2989.049100 | 64.250746 | 110.0 | **30** | 2957.783110 | 61.351625 | 127.0 |
| 700 | 14 | 8 | 3600 | 0 | 1764.227980 | 44.899458 | 27.0 | 0 | 1758.894670 | 44.698339 | 18.0 | **30** | 1733.295320 | 40.430672 | 20.0 |
| 700 | 21 | 8 | 3600 | 0 | 1002.934830 | 36.485459 | 25.5 | 0 | 1005.400970 | 36.097162 | 7.0 | **30** | 965.869280 | 31.968564 | 12.0 |
| | | Total | 78300 | 407 | 23524.928060 | — | 2596732.0 | 87 | 23607.524400 | — | 2984919.5 | 527 | 23292.533490 | — | 1082253.0 |

**Table 6** Short runtime (limited to 5 to 15 minutes) results of VNS with GH and PILOT-GRASP.

| **Instance set** | | | | **VNS with GH** | | | | **PILOT-GRASP** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|L\|$ | $\hat{t}$ [h] | $t_{max}$ [s] | #best | $\overline{obj}$ | sd | $\widetilde{g_{tot}}$ | #best | $\overline{obj}$ | sd | $\widetilde{g_{tot}}$ |
| 30 | 1 | 2 | 300 | 28 | 147.201660 | 25.480770 | 569766.5 | **29** | 147.135000 | 25.519080 | 207275.0 |
| 30 | 1 | 4 | 300 | **28** | 95.203360 | 19.886650 | 101678.5 | 24 | 95.736670 | 20.582820 | 32566.0 |
| 60 | 1 | 4 | 600 | **29** | 269.603620 | 30.419670 | 107947.0 | 18 | 271.203580 | 31.599270 | 25369.0 |
| 60 | 1 | 8 | 600 | **28** | 170.473680 | 21.976200 | 14632.0 | 16 | 171.073660 | 22.001610 | 3515.0 |
| 90 | 2 | 4 | 600 | **26** | 346.273910 | 40.604890 | 16489.5 | 6 | 349.007180 | 41.164970 | 5910.0 |
| 90 | 2 | 8 | 600 | **25** | 174.680360 | 27.952510 | 2384.5 | 6 | 176.680330 | 28.783920 | 783.0 |
| 300 | 6 | 4 | 900 | **21** | 1248.954860 | 61.250840 | 748.0 | 9 | 1250.754900 | 60.019630 | 338.5 |
| 300 | 9 | 8 | 900 | 0 | 411.991360 | 24.300130 | 35.5 | **30** | 398.524990 | 23.012590 | 29.0 |
| 500 | 10 | 4 | 900 | 0 | 2115.502050 | 70.683370 | 130.0 | **30** | 2094.169080 | 69.288040 | 80.0 |
| 500 | 15 | 8 | 900 | 0 | 692.029890 | 29.335570 | 16.0 | **30** | 666.163990 | 29.783500 | 8.0 |
| 700 | 14 | 4 | 900 | 0 | 3003.715600 | 61.370360 | 47.5 | **30** | 2962.583020 | 59.063480 | 33.0 |
| 700 | 21 | 8 | 900 | 0 | 1011.267910 | 35.516260 | 7.0 | **30** | 968.735990 | 32.905700 | 4.0 |
| | | Total: | 8400 | 185 | 9686.898260 | — | 813882.0 | 258 | 9551.768390 | — | 275910.5 |

## 10 Conclusions and Future Work

We investigated different metaheuristic approaches for finding good solutions to the problem of balancing bicycle sharing systems using a fleet of vehicles, focusing on the static problem variant where any user activities during the rebalancing process are neglected.

We started with a greedy construction heuristic for quickly generating meaningful initial solutions. Its particular feature is the greedy criterion in which an estimation of still deliverable bikes is considered. In a further step, the construction heuristic was extended to a PILOT method. The impact of different choices for the PILOT depth parameter were studied. Results showed that the unrestricted variant yielded clearly better results than any depth restriction while still performing relatively fast even on the largest instances.

To locally improve solutions, we applied a VND employing seven neighborhood structures focusing on different problem aspects. For obtaining even better solutions in longer running times, we suggested GRASP based on randomized versions of the greedy construction heuristic as well as the PILOT method and a general VNS that makes use of further, larger neighborhoods exploited by shaking.

All these local search based metaheuristics focus on the search space of vehicle routes, and corresponding loading instructions are derived for each candidate solution as second level decision variables by an embedded method. Four alternative strategies have been described for this purpose and were experimentally compared. The most general method based on linear programming and its functionally equivalent but computationally more efficient implementation based on two maximum flow calculations yield proven optimal solutions for the general non-monotonic case, where bikes may, e.g., be buffered at stations and directly be exchanged between vehicles. However, results indicate that this flexibility only rarely yields better solutions and the average quality gains are rather small. The significantly higher computational complexity does not pay off in general, and therefore we conclude to better stay with the fast and relatively simple greedy heuristic approach for determining loading instructions. Monotonicity is thus a very reasonable restriction for practical applications.

During computational experiments we compared the performance of the two construction heuristics, each optionally followed by the VND, as well as GRASP and VNS. Results show that the PILOT method yields significantly better solutions than the pure greedy construction heuristic in still relatively short running times. Furthermore, GRASP and VNS are able to significantly improve the results obtained by the construction heuristics. They ex-

hibit significant performance differences in dependence of instance size. For medium-sized instances, the VNS showed significant advantages, while for large to very large instances it was outperformed by GRASP. We conclude that for this problem GRASP scales better with respect to instance size and complexity.

In future work further performance gains might possibly be achieved by considering additional advanced neighborhood structures, e.g., very large neighborhoods based on ejection chains or mixed integer programming. For much larger BBSS instances with possibly thousands of stations, it appears to become crucial to combine the suggested methods with clustering, decomposition or multi-level optimization techniques to achieve even better scalability.

Another interesting extension would be to drop the assumption that an increase in balance is always preferred over savings in travel times and the number of loading operations. While our heuristics in principle still work with arbitrary weights for the corresponding terms in the objective function, the advanced techniques for calculating loading instructions do not necessarily produce optimal results anymore. Thus, corresponding extensions of these procedures and, in general, more advanced multi-objective optimization techniques should be considered.

Finally, it is important to also consider the dynamic (online) scenario where bikes are rented or returned by users even during the balancing process. As this problem variant introduces uncertainties, stochastic aspects need to be addressed. From the optimization point of view, it will not suffice to consider the final deviations from target balance values anymore because user demands will constantly change the fill levels of stations over time. Therefore the order in which stations are visited becomes much more relevant. In addition, it is necessary to consider user demand shifts which occur when full stations cause an increased parking position demand whereas empty stations cause an increased bike demand in the neighboring stations.

## Acknowledgments

This work is supported by the Austrian Research Promotion Agency (FFG) under contract 831740. We thank Matthias Prandtstetter, Andrea Rendl, Christian Rudloff, and Markus Straub from the Austrian Institute of Technology (AIT) for the collaboration in this project, constructive comments and for providing the data used in our test instances. In addition, we thank Citybike Wien for providing information about practical aspects of their bicycle sharing system and additional data incorporated into the test instances.

## References

1. 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), 37–61 (2011)
2. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization **10**(2), 120–146 (2013)
3. Chemla, D., Meunier, F., Pradeau, T., Calvo, R.W., Yahiaoui, H.: Self-service bike sharing systems: simulation, repositioning, pricing. Tech. Rep. hal-00824078, Centre d'Enseignement et de Recherche en Mathématiques et Calcul Scientifique (CERMICS) (2013)
4. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica **19**(4), 390–410 (1997)

5. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a dynamic public bike-sharing system. Tech. Rep. CIRRELT-2012-09, CIRRELT, Montreal, Canada (2012)
6. Dell'Amico, M., Maffioli, F., Värbrand, P.: On prize-collecting tours and the asymmetric travelling salesman problem. International Transactions in Operational Research **2**(3), 297–308 (1995)
7. DeMaio, P.: Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation **12**(4), 41–56 (2009)
8. Di Gaspero, L., Rendl, A., Urli, T.: Constraint-based approaches for balancing bike sharing systems. In: C. Schulte (ed.) Principles and Practice of Constraint Programming, *Lecture Notes in Computer Science*, vol. 8124, pp. 758–773. Springer Berlin Heidelberg (2013)
9. Di Gaspero, L., Rendl, A., Urli, T.: A hybrid ACO+CP for balancing bicycle sharing systems. In: M. Blesa, C. Blum, P. Festa, A. Roli, M. Sampels (eds.) Hybrid Metaheuristics, *Lecture Notes in Computer Science*, vol. 7919, pp. 198–212. Springer Berlin Heidelberg (2013)
10. Golden, B., Raghavan, S., Wasil, E.A. (eds.): The Vehicle Routing Problem: Latest Advances and New Challenges, *Operations Research/Computer Science Interfaces*, vol. 43. Springer (2008)
11. Hernández-Pérez, H., Salazar-González, J.J.: A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discrete Applied Mathematics **145**(1), 126–139 (2004)
12. Laporte, G., Martello, S.: The selective travelling salesman problem. Discrete Applied Mathematics **26**(2–3), 193–207 (1990)
13. Lin, J.H., Chou, T.C.: A geo-aware and VRP-based public bicycle redistribution system. International Journal of Vehicular Technology (2012)
14. Lin, J.R., Yang, T.H., Chang, Y.C.: A hub location inventory model for bicycle sharing system design: Formulation and solution. Computers & Industrial Engineering **65**(1), 77–86 (2013)
15. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research **24**(11), 1097–1100 (1997)
16. Nair, R., Miller-Hooks, E., Hampshire, R.C., Bušić, A.: Large-scale vehicle sharing systems: Analysis of Vélib'. International Journal of Sustainable Transportation **7**(1), 85–106 (2013)
17. Pfrommer, J., Warrington, J., Schildbach, G., Morari, M.: Dynamic vehicle redistribution and online price incentives in shared mobility systems. Tech. Rep. arXiv:1304.3949, Cornell University, NY (2013)
18. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In: C. Prodhon, et al. (eds.) Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing. Troyes, France (2008)
19. Raidl, G.R., Hu, B., Rainer-Harbach, M., Papazek, P.: Balancing bicycle sharing systems: Improving a VNS by efficiently determining optimal loading operations. In: M.J. Blesa, et al. (eds.) Hybrid Metaheuristics, 8th Int. Workshop, HM 2013, *LNCS*, vol. 7919, pp. 130–143. Springer (2013)
20. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In: M. Middendorf, C. Blum (eds.) Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 7832, pp. 121–132. Springer Berlin Heidelberg (2013)
21. Raviv, T., Tzur, M., Forma, I.A.: Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transportation and Logistics pp. 1–43 (2013)
22. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, pp. 219–249. Kluwer Academic Publishers (2003)
23. Rudloff, C., Lackner, B.: Modeling demand for bicycle sharing system – neighboring stations as a source for demand and a reason for structural breaks. Tech. rep., Austrian Institute of Technology, Vienna, Austria (2013)
24. Savelsbergh, M.: Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing **6**(4), 445–454 (1994)
25. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Tech. Rep. 2013-E1, Tepper School of Business, Carnegie Mellon University (2013)
26. Vansteenwegen, P., Souffriau, W., Oudheusden, D.V.: The orienteering problem: A survey. European Journal of Operational Research **209**, 1–10 (2011)
27. Voß, S., Fink, A., Duin, C.: Looking ahead with the Pilot method. Annals of Operations Research **136**, 285–302 (2005)