Balancing Bicycle Sharing Systems: An Approach for the Dynamic Case

Conference Paper · April 2014

DOI: 10.1007/978-3-662-44320-0_7

4 authors:

Christian Kloimüllner
TU Wien
11 PUBLICATIONS **111** CITATIONS
SEE PROFILE

Petrina Papazek
Central Institute for Meteorology and Geodynamics, TU Wien
9 PUBLICATIONS **198** CITATIONS
SEE PROFILE

Bin Hu
AIT Austrian Institute of Technology
45 PUBLICATIONS **420** CITATIONS
SEE PROFILE

Günther Raidl
TU Wien
245 PUBLICATIONS **4,388** CITATIONS
SEE PROFILE

Balancing Bicycle Sharing Systems: An Approach for the Dynamic Case*

Christian Kloimüllner, Petrina Papazek, Bin Hu, and Günther R. Raidl**

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

Abstract. Operators of public bicycle sharing systems (BSSs) have to regularly redistribute bikes across their stations in order to avoid them getting overly full or empty. We consider the dynamic case where this is done while the system is in use. There are two main objectives: On the one hand it is desirable to reach particular target fill levels at the end of the process so that the stations are likely to meet user demands for the upcoming day(s). On the other hand operators also want to prevent stations from running empty or full during the rebalancing process which would lead to unsatisfied customers. We extend our previous work on the static variant of the problem by introducing an efficient way to model the dynamic case as well as adapting our previous greedy and PILOT construction heuristic, variable neighborhood search and GRASP. Computational experiments are performed on instances based on real-world data from Citybike Wien, a BSS operator in Vienna, where the model for user demands is derived from historical data.

1 Introduction

Bicycle Sharing Systems (BSSs) are evolving in large cities all over the world. They offer various advantages regarding urban development, attractiveness for citizens, reduce individual motorized traffic and complement public transport. Furthermore, BSSs also contribute to public health by encouraging people to do sports [1]. A BSS consists of multiple bike stations distributed over various strategically favorable positions in the city. A registered user is allowed to rent a bike at a station and return it later at another station. Due to miscellaneous factors such as altitude of stations, demographic characteristics, or nearby public transport stops, some stations tend to run empty whereas others tend to get full. In case of an empty station, customers are not able to rent bikes while in case of a full station, customers cannot return their bikes. Therefore, BSS operators need to redistribute bikes among stations on a regular basis to avoid or at least

* This work is supported by the Austrian Research Promotion Agency (FFG), contract 831740.

** The authors thank Matthias Prandtstetter, Andrea Rendl and Markus Straub from the Austrian Institute of Technology (AIT) for the collaboration in this project.

C. Blum and G. Ochoa (Eds.): EvoCOP 2014, LNCS 8600, pp. 73–84, 2014.

minimize customer dissatisfaction. Usually this task is done by a vehicle fleet that picks up bikes from stations with excesses of bikes and delivers them to stations with deficits.

When trying to approach our definition of Balancing Bicycle Sharing System (BBSS) problem, the goal is to find a route for every vehicle with corresponding loading instructions, respectively, so that the system is brought to a balanced state and is able to fulfill user demands as much as possible. So far, almost all of our recent work considered only the static variant where it is assumed that the rebalancing process is done while the system is not in use [2–5]. This is a useful approach for BSSs which, e.g., do not operate overnight, and is also practical for strategic planning to reach desired fill levels in the long term as it depicts a simplification to the problem. In this work we extend our previous algorithms for the static case towards the dynamic scenario where we take user demands over time into account, and try to reduce unfulfilled demands during the rebalancing process as well as reaching target fill levels for stations at the end. We propose an efficient way to model and simulate these dynamics as well as adapt a greedy and PILOT construction heuristic, Variable Neighborhood Search (VNS) and GRASP accordingly.

2 Related Work

BBSS can be regarded as a special variant of the capacitated single commodity split pickup and delivery vehicle routing problem. Particular features are that we allow multiple visits of stations, consider heterogeneous vehicles, and the possibility of loading or unloading an arbitrary number of bikes.

Most related approaches address the static variant of BBSS and apply Mixed Integer Programming (MIP) techniques. A direct comparison of existing works is difficult as most of them consider different problem characteristics. Chemla et al. [6] propose a branch-and-cut algorithm on a relaxed MIP model in conjunction with a tabu search for obtaining upper bounds. However, they assume only a single vehicle and reaching the target fill levels as a hard constraint. Benchimol et al. [7] also consider a single vehicle and balance as a hard constraint and propose approximation algorithms. Raviv et al. [8] use MIP approaches with a convex penalty objective function to minimize user dissatisfaction and tour lengths for multiple vehicles. However, they ignore the number of loading operations. In our recent works we developed several metaheuristic approaches which scale well for large instances [3–5]. In particular, we introduced a Greedy Construction Heuristic (GCH) and a VNS approach in [3], a PILOT construction heuristic and GRASP in [5]. Different strategies for finding meaningful loading instructions for candidate routes including optimal ones were studied in [2]. In [4] we refined our concepts and provided more extensive computational analysis. Di Gaspero et al. [9, 10] investigate Constraint Programming (CP) approaches in conjunction with ant colony optimization, a smart branching strategy, and large neighborhood search. They test on the same BBSS variant as we do, but could not outperform our VNS approach from [3].

Concerning the dynamic BBSS scenario, there exist only few MIP approaches so far. Contardo et al. [11] present an approach utilizing Dantzig-Wolfe and Benders decomposition. They obtain upper and lower bounds for instances with up to 100 stations, but face significant gaps. Unlike our problem definition, they focus exclusively on fulfilling user demands but do not consider target fill levels. Schuijbroek et al. [12] apply a clustering-first route-second strategy. A clustered MIP heuristic, or alternatively, a CP approach handle the routing problems. In contrast to our work they define intervals for fulfilling user demands. These are considered as hard constraints whereas we try to minimize as many unfulfilled demands as possible. Additionally, they do not consider target fill levels. Chemla et al. [13] present a theoretical framework to estimate the vehicles' impacts on the system and propose heuristic approaches for a single vehicle. Besides, they suggest a pricing strategy to encourage users to return bikes at stations which tend to run empty soon. Pfrommer et al. [14] investigate a heuristic for planning tours with multiple vehicles and also suggest a dynamic pricing strategy. They periodically recompute truck tours and dynamic prices while the system is active and test with a simulation based on historic data.

Other related works examine strategic planning aspects of BSSs such as location and network design [15, 16] or system characteristics and usage patterns [17]. However, these aspects are not within the scope of this work.

3 Problem Definition

We consider the dynamic scenario of BBSS, referred to as *DBBSS*, where rebalancing is done while we simulate system usage by considering expected cumulated user demands. In addition to the input data for the static problem variant we particularly consider expected user demands from a prediction model.

For the BSS infrastructure we are given a complete directed graph $G = (V \cup \{0\}, A)$ where node set V represents rental stations, node 0 the vehicles' depot, and arc set A the fastest connection between all nodes. Each arc $(u, v) \in A$ is assigned a weight corresponding to the travel time $t_{u,v} > 0$ (including average times for loading and unloading actions). Each station $v \in V$ has a capacity $C_v \geq 0$ denoting the total number of bike slots. The initial fill level p_v is the number of available bikes at the beginning of the rebalancing while the target fill level q_v states the desired number of bikes at the end of the rebalancing. For the rebalancing procedure we are given a fleet of vehicles $L = \{1, \ldots, |L|\}$ where each vehicle $l \in L$ has a capacity $Z_l > 0$. Finally, let $\hat{t}_{\max}$ be the time budget within a vehicle has to finish its route which starts and ends at the depot 0.

Regarding user demands over time we assume the *expected cumulated demand* $\mu_v(t) \in \mathbb{R}$ occurring at each station $v \in V$ from the beginning of the rebalancing process until time t, $0 \leq t \leq \hat{t}_{\max}$ to be given as an essentially arbitrary function. The cumulated demand is calculated by subtracting the expected number of bikes to be returned from the expected number of bikes to be rent over the respective time period. An example of a demand function is shown in Figure 1. Note that we display $p_v - \mu_v(t)$ as the dash-dotted line in order to highlight the

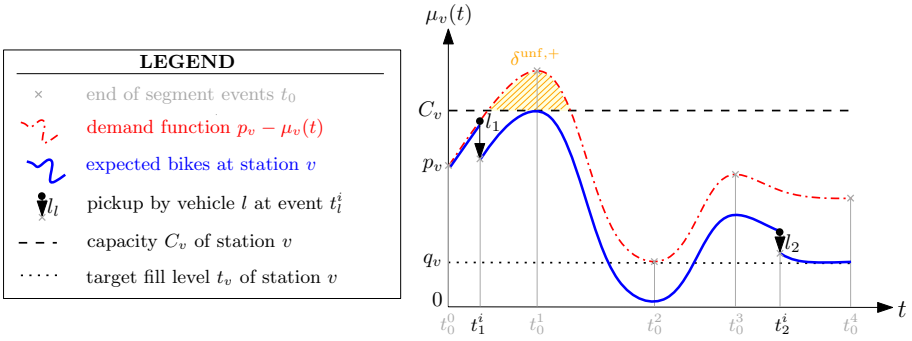

Fig. 1. Example of a demand function and two pickup events

area where unfulfilled demands occur. Thus, a positive slope of $\mu_v(t)$ indicates that more users are expected to rent bikes than to return them in the time period t, and vice versa. Demands are always fulfilled immediately as far as possible, i.e., bikes or parking slots are available. Unfulfilled demands cannot be fulfilled later and are penalized in the objective function. Let $\hat{\delta}_v^{\text{unf},-}$ denote the total amount of *unfulfilled bike demands* for station $v \in V$, i.e., the number of users who want to rent a bike but are not able to at the desired station. Analogously, let $\hat{\delta}_v^{\text{unf},+}$ refer to the total number of *unfulfilled slot demands*, i.e., the amount of users who cannot return bikes as the station is already full.

A solution to DBBSS consists of a route for every vehicle and corresponding loading instructions for every stop at a station. A route of length ρ_l is defined as an ordered, arbitrarily long sequence of stations $r_l = (r_l^1, \ldots, r_l^{\rho_l})$, $r_l^i \in V$ where the depot is assumed to be implicitly added as start and end node. The loading instruction for vehicle $l \in L$ during the i-th stop at station $v \in V$ is denoted as $y_{l,v}^i$. Positive values for $y_{l,v}^i$ denote the corresponding number of bikes to be picked up, negative values denote deliveries. Feasible solutions must fulfill the following conditions. For any station, its fill level (i.e., the number of currently available bikes) must always lie between 0 and its capacity C_v. For any vehicle $l \in L$ the load may never exceed its capacity, i.e., $b_l \leq Z_l$. Moreover, a solution is only feasible, if and only if no route's total travel time, denoted by t_l, exceeds the time budget, and additionally, every vehicle must return empty to the depot 0.

The goal is to find a route for each vehicle with corresponding loading instructions such that the following objective function is minimized:

$$f(r,y) = \omega^{\text{unf}} \sum_{v \in V} (\hat{\delta}_v^{\text{unf},-} + \hat{\delta}_v^{\text{unf},+}) + \omega^{\text{bal}} \sum_{v \in V} |q_v - p_v| + \omega^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y_{l,r_l^i}^i| + \omega^{\text{time}} \sum_{l \in L} t_l \quad (1)$$

Parameters ω^{unf}, ω^{bal}, ω^{load}, and $\omega^{\text{time}} \geq 0$ are used for controlling the relative importance of the corresponding term in the objective function. The most important goal is to minimize unfulfilled demands as well as to minimize the

deviation from the target fill levels. Secondarily, we also want to keep the total number of loading instructions and the total driving time as small as possible, however, those aspects are considered to be clearly less important.

4 Modeling the Dynamic Scenario

In this section we show how DBBSS can be modeled by calculating dynamic behavior of the system, i.e., considering the user demands, so that it can be approached by metaheuristics.

4.1 Segments and Events

One of our major aims is to avoid a time discretization of the demand functions and corresponding fill level calculations as this would introduce errors and is also time consuming if done in an appropriate resolution. Alternatively, we follow the approach of splitting each cumulated demand function into weakly monotonic segments instead of iterating through all discrete time points. Along with the practically reasonable assumption that the number of segments per station is relatively small, such an approach is much more efficient.

For this purpose, we split function $\mu_v(t)$ into monotonically weakly increasing or decreasing segments. Let $t_0 = (t_0^0, \ldots, t_0^{\rho_0})$ with $t_0^0 = 0$ be an ordered sequence of ρ_0 extreme values of $\mu_v(t)$ so that $\mu_v(t)$ is weakly monotonic for $t \in [t_0^{i-1}, t_0^i]$, $\forall i = 1, \ldots, \rho_0$, see Figure 1. Time t_0^i, $i = 1, \ldots, \rho_0$, refers to the end of the i-th weakly monotonic segment. In general, let t_l^i, $\forall l \in L$, $i = 0, \ldots, \rho_l$, be the time when vehicle l performs its i-th stop, i.e.,

$$t_l^i = \begin{cases} 0 & \text{for } i = 0 \\ t_{s_l, r_l^1} & \text{for } i = 1 \text{ if } \rho_l \geq 1 \\ t_l^{i-1} + t_{r_l^{i-1}, r_l^i} & \text{for } i = 2, \ldots, \rho_l \text{ if } \rho_l \geq 2. \end{cases} \tag{2}$$

For each station $v \in V$ we define a data structure which denotes the *series of events* $W_v = \langle (l_1, i_1), \ldots, (l_{|W_v|}, i_{|W_v|}) \rangle$. Each event (l_j, i_j), $j = 1, \ldots, |W_v|$ with $l_j \in \{0\} \cup L$ and $i_j \in \{1, \ldots, \rho_{l_j}\}$ either refers to a *station-visit event*, in which case $l_j \in L$ indicates the corresponding vehicle and i_j the number of its stop, or an *end-of-segment event*, in which case $l_j = 0$ and i_j denotes the respective segment of $\mu_v(t)$. Following the above definitions, the time of event (l_j, i_j) is $t_{l_j}^{i_j}$, and all events in W_v are ordered according to increasing times. Multiple events occurring at the same time are ordered arbitrarily, except that an end-of-segment event always appears last.

4.2 Expected Number of Bikes at Stations

For each station and event we need to derive a fill level considering the cumulated user demand as well as all performed loading or unloading instructions occurred up to this event.

Let $a_{v,j} \in [0, C_v]$ denote the expected number of bikes at station $v \in V$ and event $j = 1, \ldots, |W_v|$ by considering all expected demands fulfilled as far as possible and all pickups and deliveries performed up to and including event j. Note that, as the cumulated user demand is only a forecast model based on historical data, the fill level of every event may also be fractional. Formally, $a_{v,j}$ is calculated as follows:

$$a_{v,j} = \begin{cases} p_v & \text{for } j = 0 \\ \max(\min(a_{v,j-1} - (\mu_v(t_{l_j}^{i_j}) - \mu_v(t_{l_{j-1}}^{i_{j-1}})), C_v), 0) - y_{l_j}^{i_j} & \text{for } j = 1, \ldots, |W_v|. \end{cases} \tag{3}$$

End-of-segment events are considered for the correct computation of unfulfillable demands. For the ease of notation, the above formula considers them in the same way as *vehicle-visit events*. Since no bikes are delivered or picked up by these events, we define the loading instructions to be $y_0^i = 0$, for $i = 1, \ldots, \rho_0$.

With respect to unfulfilled demands, we distinguish between *unfulfilled bike demands* $\hat{\delta}_v^{\text{unf},-}$ and *unfulfilled slot demands* $\hat{\delta}_v^{\text{unf},+}$ for each station $v \in V$. They occur whenever the expected cumulated demand $\mu_v(t)$ over time horizon $t \in [0, \hat{t}_{\max}]$ cannot be satisfied, i.e., when $\mu_v(t) < 0 \wedge a_v(t) = 0$ or $\mu_v(t) > 0 \wedge a_v(t) = C_v$, respectively. Unfulfilled demands occurring at station v between events $j-1$ and j, $j = 1, \ldots, |W_v|$, can formally be described as

$$\delta_{v,j}^{\text{unf},-} = \max(\mu_v(t_{l_j}^{i_j}) - \mu_v(t_{l_{j-1}}^{i_{j-1}}) - a_{v,j-1} + y_{l_j}^{i_j}, 0) \tag{4}$$

$$\delta_{v,j}^{\text{unf},+} = \max(-(\mu_v(t_{l_j}^{i_j}) - \mu_v(t_{l_{j-1}}^{i_{j-1}})) - (C_v - a_{v,j-1}) - y_{l_j}^{i_j}, 0), \tag{5}$$

and consequently the overall unfulfilled demands are

$$\hat{\delta}_v^{\text{unf},-} = \sum_{j=1}^{|W_v|} \delta_{v,j}^{\text{unf},-}, \quad \text{and} \quad \hat{\delta}_v^{\text{unf},+} = \sum_{j=1}^{|W_v|} \delta_{v,j}^{\text{unf},+}. \tag{6}$$

4.3 Classification of Stations

We assume that the stations are well-designed in a sense that their capacities are sufficiently large for daily fluctuations, i.e., it will not be necessary to pick up and deliver bikes to the same station at different times on a single day in order to fulfill all demands. Furthermore, we have shown in previous work [3] that the monotonicity restriction (i.e., it is allowed to either only pick up or deliver bikes at a station) has in practice only a neglectably small impact on the solution quality but substantially simplifies the problem. Additionally, our project partner Citybike Wien mentioned that they only perform either pickups or deliveries at a particular station on the same day. Therefore, we again classify the stations into pickup stations $V_{\text{pic}} \subseteq V$ and delivery stations $V_{\text{del}} \subseteq V$ and impose monotonicity, i.e., allow only the respective operations.

In the static case this classification is done by considering the total deviation in balance for a particular station, i.e., $p_v - q_v \ \forall v \in V$. If this value is less than 0, then the corresponding station refers to the set of delivery stations, and

otherwise it is classified as a pickup station. However, in the dynamic case we have to consider user demands during the rebalancing process along with the scaling factors in the objective function (1).

Thus, we consider the situation when no rebalancing is done at all. Based on equation (6) and objective function (1) we determine for each station $v \in V$ the total penalty for slot deficit and bike deficit:

$$\begin{aligned}\delta_v^{\text{missing}} = \omega^{\text{unf}}\hat{\delta}^{\text{unf},+} + \omega^{\text{bal}}\min(0, a_{v,|W_v|} - q_v) - \\ \omega^{\text{unf}}\hat{\delta}^{\text{unf},-} - \omega^{\text{bal}}\min(0, q_v - a_{v,|W_v|}).\end{aligned} \tag{7}$$

If $\delta_v^{\text{missing}} < 0$, v is a delivery station. If $\delta_v^{\text{missing}} > 0$, v is a pickup station. Otherwise, if $\delta_v^{\text{missing}} = 0$, the station is already balanced, and thus, will not be considered anymore.

4.4 Restrictions on Loading Instructions

For every stop of a vehicle, we need to calculate how many bikes the vehicle is allowed to pick up or deliver at most so that the capacity constraints are never violated and unnecessary unfulfilled demands are never introduced. These bounds are then utilized to set loading instructions for the corresponding vehicle stops later during the optimization process. Formally, we define slacks $\Delta y_{v,j}^-$ and $\Delta y_{v,j}^+$ as the maximum amount of bikes which may be delivered/picked up at station v and event $j = 1, \ldots, |W_v|$.

$$\Delta y_{v,j}^- = \begin{cases}\max(0, q_v - a_{v,|W_v|}) & \text{for } j = |W_v| \\ \min(C_v - a_{v,j}, \Delta y_{v,y+1} + \delta_{v,j+1}^{\text{unf},-}) & \text{for } j = 1, \ldots, |W_v| - 1\end{cases} \tag{8}$$

$$\Delta y_{v,j}^+ = \begin{cases}\max(0, a_{v,|W_v|} - q_v) & \text{for } j = |W_v| \\ \min(a_{v,j}, \Delta y_{v,j+1} + \delta_{v,j+1}^{\text{unf},+}) & \text{for } j = 1, \ldots, |W_v| - 1.\end{cases} \tag{9}$$

Note, that we have to iterate backwards by starting with the last event until we reach the time when the currently considered vehicle stop occurs.

By definition, let $\Delta y_{v,j}^{\text{unf},+}$ and $\Delta y_{v,j}^{\text{unf},-}$ denote the slack without including the last event, i.e., starting with event $j = |W_v| - 1$. These two terms are used by the construction heuristic in the next section.

5 Greedy Construction Heuristic

We extend the Greedy Construction Heuristic (GCH) from our previous work [3] to fit the dynamic case. A vehicle tour is built by iteratively appending stations from a set of *feasible successors* $F \subseteq V$. This set includes each station which can be *improved* and is reachable within the vehicles time budget. An improvement may be achieved if $\delta_v^{\text{missing}} > 0$, $\forall v \in V_{\text{pic}}$, or $\delta_v^{\text{missing}} < 0$, $\forall v \in V_{\text{del}}$. Then, for each station $v \in F$ we compute the total number of bikes that can either be picked up from or delivered to this station:

$$\gamma_v = \begin{cases}\min(\Delta y_{v,j}^-, Z_l - b_l) & \text{for } v \in F \cap V_{\text{pic}}, \\ \min(\Delta y_{v,j}^+, b_l) & \text{for } v \in F \cap V_{\text{del}}.\end{cases} \tag{10}$$

Note that b_l denotes the number of bikes currently stored in vehicle l. As shown in (10), we need the slacks for determining possible loading instructions as they are calculated by equation (8). In order to guarantee that vehicles return empty to the depot, we correct the load for pickup stations by estimating the amount of bikes that can be delivered afterwards in the same fashion as in [3] by recursively looking forward.

It is necessary to consider impacts of loading instructions on a station with respect to target fill level and unfulfilled demands separately and weight them with ω_{bal} and ω_{unf} in the same way as it is done in the objective function (1). We obtain

$$g(v) = \begin{cases} \frac{\omega_{\text{bal}} \cdot \min(\gamma_v, \max(0, a_{v,|W_v|} - q_v)) + \omega_{\text{unf}} \cdot \min(\gamma_v, \Delta y_{v,j}^{\text{unf},+})}{t_{u,v}} & \forall v \in V_{\text{pic}}, \\ \frac{\omega_{\text{bal}} \cdot \min(\gamma_v, \max(0, q_v - a_{v,|W_v|})) + \omega_{\text{unf}} \cdot \min(\gamma_v, \Delta y_{v,j}^{\text{unf},-})}{t_{u,v}} & \forall v \in V_{\text{del}}, \end{cases} \tag{11}$$

where $t_{u,v}$ is the travel time from the vehicle's last stop u to station v. In each greedy iteration the station with the highest $g(v)$ is appended to the currently considered vehicle tour. Loading instructions are set as follows:

$$y_{v,j} = \gamma_v \text{ if } v \in V_{\text{pic}}, \quad \text{and} \quad y_{v,j} = -\gamma_v \text{ if } v \in V_{\text{del}} \tag{12}$$

Since in the dynamic case timing is important, we additionally introduce a term which we refer to as *urgency*. It states how urgent it is to visit stations with future unfulfilled demands. We propose two methods to compute this value.

Additive Urgency: For a station v we consider the time of the next period where unfulfilled demands occur. If the vehicle cannot reach the station until the first period starts, we consider the next period, and so on. In case a station has no periods of unfulfilled demands at all or none of them are reachable in time, it is ignored. Moreover, we introduce an additional scaling factor ω_{urg} which denotes the importance of urgency. Formally,

$$u_{\text{add}} = \begin{cases} 0 & \text{if } \; t_v^{\text{unf}} < t_{u,v} \\ \omega_{\text{urg}} \cdot \frac{\delta_{\text{unf}}}{t_v^{\text{unf}}} & \text{if } \; t_v^{\text{unf}} \geq t_{u,v} \end{cases} \tag{13}$$

where t_v^{unf} is the time left up to the start of the next unfulfilled demand for station $v \in V$ and $t_{u,v}$ is the travel time to the considered station which, by definition, has to be greater than 0. The greedy value including the urgency of the visit $g'(v)$ is then calculated as $g'(v) = g(v) + u_{\text{add}}$.

Multiplicative Urgency: In the multiplicative approach we multiply the basic $g(v)$ from (11) by an *exponential function*. Again, we consider the time until the next unfulfilled demand, analogously as for the additive approach. The term is computed as

$$u_{\text{mul}} = \exp(-\max(0, t_v^{\text{unf}} - t_{u,v}) \cdot \omega_{\text{urg}}). \tag{14}$$

The greedy value criterion is then extended to $g'(v) = g(v) \cdot u_{\text{mul}}$.

PILOT Construction Heuristic: Due to the nature of greedy algorithms, shortsighted decisions cannot be completely avoided no matter how we choose the greedy evaluation criterion. Therefore, we use the PILOT method [18] to address this drawback. The functionality remains the same as in [5] which extends GCH by evaluating each potential successor in a deeper way by constructing a complete temporary route from it, and finally considering its objective value as $g(v)$.

6 Metaheuristic Approaches

In order to further improve the results obtained by the construction heuristics, we apply Greedy Randomized Adaptive Search (GRASP) and Variable Neighborhood Search (VNS). For both metaheuristic approaches we use an incomplete solution representation based on storing for each vehicle $l \in L$ its route $r_l = (r_l^1, \ldots, r_l^{\rho_l})$ only. The loading instructions $y_{l,v}^i,\ l \in L,\ v \in V,\ i = 1, \ldots, \rho_l$ are efficiently calculated during evaluation by applying the same greedy strategy as in GCH, see Section 5, utilizing the restriction procedure from Section 4.4 to obtain bounds on the y-variables and accelerate the calculations.

Variable Neighborhood Search: The VNS approach from [3] is adapted with respect to the procedure for deriving loading instructions. The general layout and neighborhood structures remain the same. We use remove-station, insert-unbalanced-station, intra-route-2-opt, replace-station, intra-or-opt, 2-opt*-inter-route-exchange and intra-route 3-opt neighborhood structures for local improvement within an embedded Variable Neighborhood Descent (VND), while for shaking we apply move-sequence, exchange-sequence, destroy-&-recreate, and remove-stations operations.

Greedy Randomized Adaptive Search: We also consider GRASP by extending the construction heuristics in the same way as in our previous work [5] with adaptations for the dynamic problem variant. The idea is to iteratively apply GCH or PILOT from Section 5 and locally improve each solution with VND. While there we always select the best successor, we use for GRASP a restricted candidate list with respect to the greedy evaluation criterion. The degree of randomization is controlled by a parameter $\alpha \in [0, 1]$. In the dynamic case we used the same values which turned out to work best in the static case.

7 Computational Results

We performed comprehensive tests for our DBBSS approaches. Generating new benchmark instances was necessary in order to introduce the user demand values. They are based on the same set of Vienna's real Citybike stations we used in our previous works [2–5]. Cumulated user demands for the stations are piecewise linear functions derived from historical data based on an hourly discretization. The instances we use in this paper contain 30 to 90 stations with different numbers of vehicles and different time budgets and are available at[1]. As the BSS in

[1] `https://www.ads.tuwien.ac.at/w/Research/Problem_Instances#bbss`

Table 1. Results of GCH, PILOT, and the variants with VND

Inst. set			GCH				PILOT				GCH-VND				PILOT-VND			
$\|V\|$	$\|L\|$	$\hat{t}_{\max}$	#best	$\overline{\text{obj}}$	sd	$\widetilde{\text{t}_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{\text{t}_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{\text{t}_{\text{tot}}}$ [s]	#best	$\overline{\text{obj}}$	sd	$\widetilde{\text{t}_{\text{tot}}}$ [s]
30	1	8h	0	54.06	12.50	< 0.1	0	50.98	11.19	0.1	**18**	50.61	11.56	0.4	13	49.97	10.95	0.4
30	2	4h	0	59.79	15.65	< 0.1	1	55.47	13.78	< 0.1	9	55.87	13.58	0.2	**22**	54.89	13.44	0.1
60	1	8h	0	186.49	28.14	< 0.1	1	180.59	28.81	0.5	8	180.20	28.72	0.5	**23**	178.85	29.29	0.9
60	2	4h	0	202.69	31.82	< 0.1	0	191.06	29.98	0.2	9	193.32	30.06	0.4	**21**	189.69	29.81	0.4
60	2	8h	0	104.49	12.77	< 0.1	0	98.64	11.03	0.9	12	98.00	12.05	2.4	**18**	96.74	10.80	3.2
60	4	4h	0	118.76	17.38	< 0.1	0	106.98	13.53	0.4	4	108.96	13.34	1.8	**26**	105.36	13.41	1.3
90	1	8h	0	354.83	34.79	< 0.1	0	346.73	33.49	1.1	6	348.20	34.74	0.7	**24**	344.99	33.45	1.5
90	2	4h	0	371.13	34.55	< 0.1	0	360.49	36.06	0.5	5	362.74	35.53	0.5	**25**	358.21	35.09	0.9
90	2	8h	0	232.86	27.07	< 0.1	0	221.02	24.24	2.1	3	223.16	24.71	2.6	**27**	218.57	23.86	4.1
90	3	8h	0	155.26	19.27	< 0.1	0	144.35	16.80	2.8	6	144.43	17.94	8.6	**24**	141.25	16.26	6.9
90	4	4h	0	254.25	27.51	< 0.1	0	239.70	27.86	1.0	7	242.91	27.90	2.1	**23**	237.47	27.63	2.2
90	5	4h	0	210.12	24.26	< 0.1	0	194.03	24.55	1.2	7	195.75	24.38	4.2	**23**	191.72	23.96	3.3
		Total	0	2304.73	285.72	< 0.1	2	2190.04	271.31	10.8	94	2204.15	274.48	24.4	269	2167.71	267.95	25.2

Table 2. Results of static VNS, dynamic VNS, and PILOT-GRASP

Inst. set			SVNS			DVNS			PILOT-GRASP		
$\|V\|$	$\|L\|$	$\hat{t}_{\max}$	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd	#best	$\overline{\text{obj}}$	sd
30	1	8h	4	54.90	10.93	**20**	47.36	10.51	9	47.54	10.57
30	2	4h	4	58.68	13.08	**19**	50.84	11.50	11	50.84	11.35
60	1	8h	0	197.25	30.01	**29**	172.30	27.13	4	172.84	26.96
60	2	4h	0	207.09	30.37	**22**	182.12	29.28	10	183.09	29.13
60	2	8h	0	114.64	12.75	**26**	91.80	10.63	4	92.30	10.40
60	4	4h	0	126.42	15.11	**23**	99.24	11.39	7	99.85	11.54
90	1	8h	0	368.18	37.47	**19**	337.92	32.74	11	338.49	32.23
90	2	4h	0	380.50	38.80	**19**	349.98	33.97	11	351.05	34.74
90	2	8h	0	242.60	26.09	11	210.62	23.79	**19**	210.19	23.03
90	3	8h	0	168.99	16.31	11	135.97	15.10	**19**	135.40	15.06
90	4	4h	0	262.41	30.41	**17**	225.94	25.77	13	226.39	26.19
90	5	4h	0	216.53	23.76	**18**	182.03	21.82	12	182.20	22.21
		Total	8	2398.19	285.10	234	2086.11	253.63	130	2090.16	253.38

Vienna currently consists of 111 stations and 1300 bicycles the instances used inhere are realistic and praxis-relevant. For each parameter combination exists a set of 30 independent instances. All our algorithms are implemented 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. The scaling factors of the objective function are set to $\omega^{\text{unf}} = \omega^{\text{bal}} = 1$, $\omega^{\text{load}} = \omega^{\text{time}} = \frac{1}{100\,000}$, i.e., an improvement with respect to balance and/or unfulfilled demands is always preferred over reducing the tour length and/or the number of loading instructions. For the greedy evaluation criterion we use multiplicative urgency. We omit a detailed comparison since the difference between these strategies becomes more significant only on larger instances with hundreds of stations.

In Table 1 we compare different methods for quickly obtaining good starting solutions, namely GCH, PILOT, GCH with VND, and PILOT with VND. The columns show the instance characteristics, and for each algorithm the number of times the corresponding approach yields the best result (#best), the average objective values ($\overline{\text{obj}}$), the standard deviations (sd), and the average CPU-times $\widetilde{\text{t}_{\text{tot}}}$. The differences of the average objective values are frequently relatively small due to the weight factors ω^{load} and ω^{time}, but they are still crucial for evaluating the quality of solutions. Therefore, the #best numbers give us a better indication

of which algorithm variants perform best. We observe that PILOT outperforms GCH on every instance while the additional time is only moderate. This trend continues when we add VND to further improve the solutions. Not only does PILOT-VND outperform GCH-VND, but it also requires less time. This is due to the superior starting solutions, so VND terminates after fewer iterations.

In Table 2 we test our metaheuristic approaches and additionally compare them to the VNS for the static case from [3], denoted by SVNS. For a reasonable comparison, SVNS initially converts a DBBSS instance into a static one by adding for each station the final cumulative user demand to the respective target value; negative values and values exceeding the station capacity C_v are replaced by zero and C_v, respectively. The idea is to neglect the timing aspects of station visits and check if this static VNS is able to find reasonable solutions also for the dynamic case. To assure always obtaining feasible solutions to DBBSS in the end, loading instructions for the finally best static solution are recalculated by the new greedy strategy of the dynamic case. We observe that GCH from Table 1 already performs a little bit better than the SVNS. DVNS and GRASP are able to compute results that are more than 10% better than those of SVNS. Therefore, we conclude that although it is possible to apply algorithms for the static case to the dynamic scenario, dedicated dynamic approaches taking time-dependent user demands into account are clearly superior. Among the dynamic approaches DVNS performs best on most of the considered instances. According to a Wilcoxon signed-rank test, all observed differences on the overall number of best solutions among any pair of compared approaches are statistically significant with an error level of less than 1%.

8 Conclusions and Future Work

In this work we showed how to extend the metaheuristics developed in our previous work for static BBSS to the significantly more complex dynamic variant. Starting from a model which can handle essentially arbitrary time-dependent expected user demand functions, we proposed an efficient way to calculate loading instructions for vehicle tours. We use an objective function where the weights of unfulfilled user demands and target fill levels can be adjusted in an easy way. Practically, this has a high relevance for the BSS operator. We also extended our previously introduced construction heuristics, VNS and GRASP, so that dynamic user demands are considered appropriately. Tests on practically realistic instances show that the dynamic approaches indeed make sense. Depending on the available time for optimization, greedy or PILOT construction heuristics are useful for fast runs, while VNS is most powerful for longer runs.

In future work it would be particularly interesting to also consider the impact of demand shifts among stations when their neighbors become either full or empty. Especially, when users want to return bikes and an intended target station is full, this demand obviously will not diminish but be shifted to some neighboring station(s). Considering this aspect might lead to an even more precise model, but also increases the model's complexity significantly.

References

1. DeMaio, P.: Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation 12(4), 41–56 (2009)
2. Raidl, G.R., Hu, B., Rainer-Harbach, M., Papazek, P.: Balancing bicycle sharing systems: Improving a VNS by efficiently determining optimal loading operations. In: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (eds.) HM 2013. LNCS, vol. 7919, pp. 130–143. Springer, Heidelberg (2013)
3. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In: Middendorf, M., Blum, C. (eds.) EvoCOP 2013. LNCS, vol. 7832, pp. 121–132. Springer, Heidelberg (2013)
4. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. Technical Report TR 186-1-13-01, Vienna, Austria (29 pages, 2013, submitted to the JOGO)
5. Papazek, P., Raidl, G.R., Rainer-Harbach, M., Hu, B.: A PILOT/VND/GRASP hybrid for the static balancing of public bicycle sharing systems. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST. LNCS, vol. 8111, pp. 372–379. Springer, Heidelberg (2013)
6. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization 10(2), 120–146 (2013)
7. 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)
8. Raviv, T., Tzur, M., Forma, I.A.: Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transp. and Log., 1–43 (2013)
9. Di Gaspero, L., Rendl, A., Urli, T.: A hybrid ACO+CP for balancing bicycle sharing systems. In: Blesa, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (eds.) HM 2013. LNCS, vol. 7919, pp. 198–212. Springer, Heidelberg (2013)
10. Di Gaspero, L., Rendl, A., Urli, T.: Constraint-based approaches for balancing bike sharing systems. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 758–773. Springer, Heidelberg (2013)
11. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a dynamic public bike-sharing system. Technical Report CIRRELT-2012-09, Montreal, Canada (2012)
12. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Technical Report 2013-E1, Tepper School of Business, Carnegie Mellon University (2013)
13. Chemla, D., Meunier, F., Pradeau, T., Calvo, R.W., Yahiaoui, H.: Self-service bike sharing systems: simulation, repositioning, pricing. Technical Report hal-00824078, CERMICS (2013)
14. Pfrommer, J., Warrington, J., Schildbach, G., Morari, M.: Dynamic vehicle redistribution and online price incentives in shared mobility systems. Technical report, Cornell University, NY (2013)
15. Lin, J.H., Chou, T.C.: A geo-aware and VRP-based public bicycle redistribution system. International Journal of Vehicular Technology (2012)
16. 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)
17. Nair, R., Miller-Hooks, E., Hampshire, R.C., Bušić, A.: Large-scale vehicle sharing systems: Analysis of Vélib'. Int. Journal of Sustain. Transp. 7(1), 85–106 (2013)
18. Voß, S., Fink, A., Duin, C.: Looking ahead with the PILOT method. Annals of Operations Research 136, 285–302 (2005)