

Constraint-based approaches for Balancing Bike Sharing Systems

Luca Di Gaspero¹, Andrea Rendl², and Tommaso Urli¹

¹ Scheduling and Timetabling Group,
Department of Electrical, Management and Mechanical Engineering,
University of Udine,
Via Delle Scienze, 206 - 33100 Udine, Italy
{luca.digaspero|tommaso.urli}@uniud.it

² Dynamic Transportation Systems,
Mobility Department,
Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
andrea.rendl@ait.ac.at

Abstract. In order to meet the users' demand, bike sharing systems must be regularly rebalanced. The problem of balancing bike sharing systems (BBSS) is concerned with designing optimal tours and operating instructions for relocating bikes among stations to maximally comply with the expected future bike demands. In this paper, we tackle the BBSS by means of Constraint Programming: first, we introduce two novel constraint models for the BBSS including a *smart* branching strategy that focusses on the most promising routes. Second, in order to speed-up the search process, we incorporate both models in a Large Neighborhood Search (LNS) approach that is adapted to the respective CP model. Third, we perform an extensive computational evaluation on instances based on real-world data, where we see that the LNS approach outperforms the Branch & Bound approach and is competitive with other existing approaches.

1 Introduction

Bike sharing systems are a very popular means to provide bikes to citizens in a simple and cheap way. The idea is to install bike stations at various points in the city, from which a registered user can easily loan a bike by removing it from a specialized rack. After the ride, the user may return the bike at any arbitrary station (if there is a free rack). This service is mainly public or semi-public, often initiated to increase the attractiveness of non-motorized means of transportation and is typically almost free of charge for the users. This, among other reasons, is why bike sharing systems have become particularly popular and an essential service in many European cities.

Depending on their location, bike stations have specific patterns regarding when they are empty or full. For instance, in cities where most jobs are located

near the city centre, the commuters cause certain peaks in the morning: the central bike stations are filled, while the stations in the outskirts are emptied. Furthermore, stations located on top of a hilly region are more likely to be empty, since users are less keen on cycling up a hill and thus less keen on returning a bike to such a station. These differences in flows are one of several reasons why many stations have extremely high or low bike loads over time, which often causes difficulties: on the one hand, if a station is empty, users cannot loan bikes from it, thus the demand cannot be met by the station. On the other hand, if a station is full, users cannot return bikes and have to find alternative stations that are not yet full. These issues can result in substantial user dissatisfaction which may eventually lead users to abandon the service. This is why nowadays most bike sharing system providers take measures to *rebalance* them.

Balancing a bike sharing system is typically done by employing a fleet of trucks that move bikes between unbalanced stations overnight. More specifically, each truck starts from a depot and travels from station to station in a tour, performing loading instructions (adding or removing bikes) at each stop. After servicing the last station, the empty truck returns to the depot.

Finding optimal tours and loading instructions is a challenging task: the problem consists of a touring problem that is combined with the problem of distributing single-commodities (bikes) to meet the demand. Furthermore, since most bike sharing systems typically have a large number of stations (≥ 100), but a small fleet of trucks, the trucks can only service a subset of unbalanced stations in a reasonable time, thus it is also necessary to decide *which* stations should be balanced.

In this work, we tackle the problem of balancing bike sharing systems in two steps. First, we formulate the problem as two different CP models: a *routing model* based on the classical Vehicle Routing Problem (VRP) formulation, and a *step model* that involves a planning perspective of the problem. We discuss both models in detail and compare their performance in a computational evaluation. In a second step, we employ each CP model in a Large Neighborhood Search (LNS) approach that is customized according to the features of the respective CP model.

This paper is structured as follows: Section 2 gives a detailed problem description of balancing bike sharing systems, including our notation. Section 3 introduces our two CP formulations for the BBSS: the routing model in Sec. 3.1, and the step model in Sec. 3.2. Then we discuss our LNS approach in Section 4 and summarize our computational evaluation in Section 5. Section 6 concludes the paper.

1.1 Related Work

Balancing of bike sharing systems has become an increasingly studied problem in the last few years. Benchimol et al. [1] consider the rebalancing as hard constraint and the objective is to minimize the travel time. They study different approximation algorithms on various instance types and derive different approximation

factors for certain instance properties. Furthermore, they present a branch-and-cut approach based on an ILP including subtour elimination constraints. Concardo et al. [5] consider the dynamic variant of the problem and present a MIP model and an alternative Dantzig-Wolfe decomposition and Benders decomposition method to tackle larger instances. Raviv et al. [10] present two different MILP formulations for the static BBSP and also consider the stochastic and dynamic factors of the demand. In the approach of Chemla et al. [4], a branch-and-cut approach based on a relaxed MIP model is used in combination with a tabu search that provides upper bounds. Rainer-Harbach et al. [9] propose a heuristic approach for the BBSP in which effective routes are calculated by a variable neighbourhood search (VNS) metaheuristic and the loading instructions are computed by a helper algorithm, where they study three different alternatives (exact and heuristic) as helper algorithms. Schuijbroek et al. [12] propose a new cluster-first route-second heuristic, in which the clustering problem simultaneously considers the service level feasibility constraints and approximate routing costs. Furthermore, they present a constraint programming model for the BBSP that is based on a scheduling formulation of the problem and therefore differs significantly from our formulations.

In [6] we presented a hybrid approach for the BBSS by combining CP with Ant Colony Optimization (ACO). In that work, we introduced a VRP-based CP formulation and integrate ACO into its search procedure to improve its performance. This work extends our previous work by introducing a novel CP model that is inspired by AI-planning. Moreover we revise the VRP-based model presented in [6] and we develop a problem-specific branching strategy. Finally, in this paper we attempt to enhance the two models by combining them into Large Neighbourhood Search (LNS).

2 Balancing Bike Sharing Systems

The problem of balancing a bike sharing system (BBSS) is concerned with finding *tours* for a fleet of vehicles and the respective *loading instructions* per stop such that the bike sharing system is maximally balanced after the vehicles finish their tour. Note, that we consider the static case of the BBSS where we assume that no bikes are moved independently between stations during the balancing operation (in other words, we assume that there are no customers using the service during balancing which can be a valid approximation for balancing systems at night).

Bike sharing systems consist of bike stations $\mathcal{S} = \{1, \dots, S\}$ that are distributed all over the respective city. Each station $s \in \mathcal{S}$ has a maximal capacity of C_s bike racks and holds b_s bikes where $0 \leq b_s \leq C_s$. The target value t_s for station s states how many bikes the station should ideally hold to satisfy the customer demand. The values for t_s are derived in advance from a user demand model where $0 \leq t_s \leq C_s$. Please note, that depending on the demand, stations are considered as either ‘sink’ or ‘source’ stations. This means that bikes cannot be removed from ‘sink’ stations, and bikes cannot be added to ‘source’ stations.

A fleet of vehicles $\mathcal{V} = \{1, \dots, V\}$ with capacity $c_v > 0$ and initial load $\hat{b}_v \geq 0$ for each vehicle $v \in \mathcal{V}$, move bikes between stations to reach the stations' target values. The vehicles are associated with depot D where they start and end their tour. Thus, the set of possible stops in a tour is denoted $\mathcal{S}_d = \mathcal{S} \cup D$. We have a time budget of $\hat{t} > 0$ time units to complete the balancing operation (and after which every vehicle has to have reached the depot). The travel times between all possible stops are given by the matrix $travelTime_{u,v}$ where $u, v \in \mathcal{S}_d$, which includes an estimate of the processing times needed to serve the station, if $v \in \mathcal{S}$.

The goal is to find a tour for each vehicle including loading instructions for each visited station. The loading instructions state how many bikes have to be removed from, or added to the station, respectively. Naturally, the loading instructions must respect the maximal capacity and current load of both the vehicle and the station. Furthermore, each vehicle can only operate within the overall time budget and has to distribute all loaded bikes before returning to the depot (i.e., the truck has to be empty when returning to the depot).

After every vehicle has returned to the depot, each station $s \in \mathcal{S}$ has a new load of bikes, denoted b'_s . Obviously, the closer b'_s is to the the desired target value t_s , the better the solution. Thus, our objective is to find tours that manipulate the station states such that they are as close as possible to their target values. Furthermore, we are interesting in finding a low-cost route r_v for each vehicle $v \in \mathcal{V}$, so we also minimize the total travel time (which is equivalent to minimize the total traveling cost).

Therefore, we introduce an objective function f that contains two components: first, the sum of the deviation of b'_s from t_s over all stations $s \in \mathcal{S}$, and second the travel time for each vehicle:

$$f(\sigma) := w_1 \sum_{s \in \mathcal{S}} |b'_s - t_s| + w_2 \sum_{v \in \mathcal{V}} \sum_{(u,w) \in r_v} travelTime_{u,w} \quad (1)$$

Note that Equation 1 defines a scalarization over a naturally multi-objective problem. Some points in the Pareto optimal set are hence neglected by construction. Our main reason for this choice is the need to compare with the current best approaches [9], which employ an equivalent scalarization. Furthermore, to the best of our knowledge, multi-objective propagation techniques are still a relatively unexplored research area.

3 Constraint Models for the BBSS

In this section we present two constraint formulations for the Balancing Bike Sharing Systems problem (BBSS).

3.1 Routing Model

The routing model is an adaption of the constraint model of the classical vehicle routing problem (VRP) that is described in [8]. The routing model uses successor variables ($succ_i$) to model the path of each vehicle and service variables ($service_i$) to represent the operations at each station.

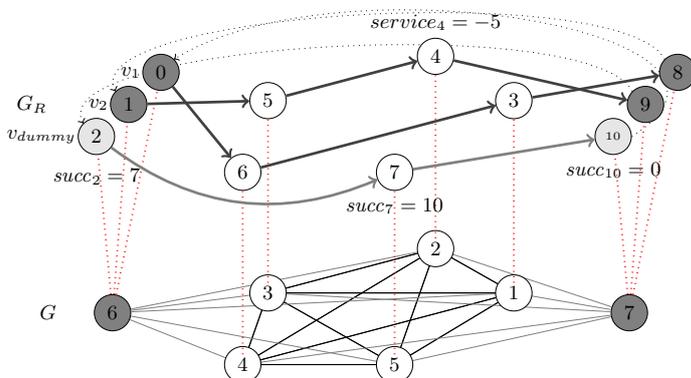


Fig. 1. Graph encoding of the BBSS problem employed in the routing CP model. The lower layer shows the original graph, whereas the upper layer shows the encoded graph, in the case of two vehicles, and the edges selected in a possible solution. The sub-path starting at node 2 and ending at node 10 (i.e., the dummy vehicle) corresponds to the set of unserved nodes.

The essence of the graph encoding underlying the routing model is depicted in Figure 1 along with the elements of a possible solution. According to the formulation proposed in [8], the graph structure of the original problem G (lower layer) is encoded into an extended graph G_R (upper layer) by considering one replicate of the starting depot for each vehicle in order to identify each vehicle route as a sub-path in the graph starting at that node. The successor of the ending depot for a given vehicle is set to be the starting depot of the following vehicle (modulo the number of vehicles) so that we are searching for an Hamiltonian circuit in the extended graph. Moreover, for modeling service optionality, there is an additional vehicle v_{dummy} , whose sub-path comprises all the stations that are left unserved.

For the sake of brevity we do not give here a detailed specification of the variables and constraints that are involved in the model, which can be found in [6]. The main difference with respect to the cited paper, however, is the search strategy which will be outlined in the following section. Another difference is that the current version of the routing model employs the Hamiltonian `circuit` global constraint instead of the `alldifferent` on the `succ` variables. Although the two formulations are equivalent, the `circuit` variant has a better sub-tour elimination behavior.

As a final remark, it is important to notice that the basic model does not allow revisiting the same station more than once. Nevertheless, this limitation can be dropped by replicating each station in the extended graph G_R according to the number of revisits that will be allowed.

Search strategy. The search strategy for the routing model attempts to incrementally construct the route for each vehicle by considering together the *succ* and the *service* variables, and employing a *smart* branching heuristic.

In detail, given a partial route for a vehicle, the next variable to be selected is the *succ* variable of the last node in the route. The possible values for this variable (i.e., the next station to be served) are ordered according to the contribution of the current vehicle load and the possible service at the next station in reducing its unbalance by preferring those that have a higher impact in this reduction. This value selection heuristic performs a one step look-ahead toward the next variable to be selected. Once the *succ* variable is set, the *service* variable of the next station is selected for branching. To be consistent with the look-ahead, the possible values for this variable are ordered according to their contribution in reducing the unbalance.

When the current route has to be ended because the time budget of the vehicle is finished or we have reached the vehicle’s final depot, the next station to be considered for branching is the *succ* variable selection is the starting depot of the following vehicle.

Model extensions. The model, and the solution methods built upon it, is quite flexible and make it possible to easily incorporate additional real-world aspects, which are not considered in the current problem statement.

For example it is quite trivial to allow for waiting times at stations (e.g., for avoiding the contemporary presence of two vehicles at the same station), by relaxing an equality constraint in the time accumulation formula. Also loading times at stations can be straightforwardly taken into account in the model.

Other possible extensions to the model include allowing the use of stations as *intermediate* depots by neglecting the ‘sink’ and ‘source’ concept or the relaxation of the constraint that requires the vehicle to be empty at the end of the route.

Finally, the model can be immediately adapted to consider the related problem of minimizing the working times in case of full rebalancing (similarly to [1]), just by imposing that the final unbalance of the stations should be zero.

3.2 Step Model

The step model considers the problem as a planning problem with a planning horizon of K steps, i.e., we try to find a route (with respective loading instructions) of maximal length K for each vehicle, where the first and the last stop is the depot. We introduce the set of steps $\mathcal{K} = \{0, \dots, K\}$ where 0 is the initial state and step K is the final state, thus each vehicle visits $K - 1$ stations. We set K to an estimated upper bound

$$K = \left\lceil \frac{\hat{t}}{\tilde{t}} \right\rceil + 1 \quad (2)$$

where \tilde{t} is the median of all travel times.

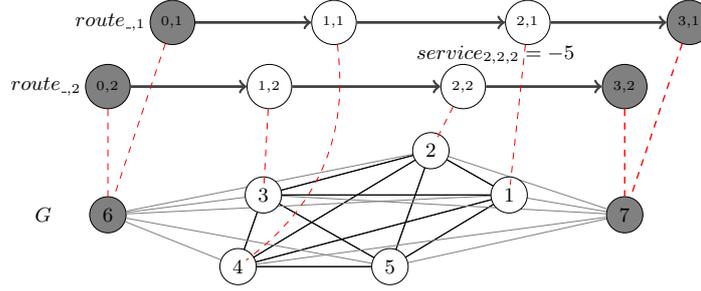


Fig. 2. Solution representation for the step CP model. The lower layer shows the original graph, whereas the upper layer shows the decision variables of the step model, i.e., the routes variables for two vehicles, and an example of the service variables.

name	[dimension]	domain	description
$route$	$[\mathcal{K}][\mathcal{V}]$	\mathcal{S}_D	stop of vehicle $v \in \mathcal{V}$ at step $k \in \mathcal{K}$
$service$	$[\mathcal{K}][\mathcal{S}][\mathcal{V}]$	$\{-C_{max}, C_{max}\}$	removed/added bikes at station $s \in \mathcal{S}$ by vehicle $v \in \mathcal{V}$ at step $k \in \mathcal{K}$
$activity$	$[\mathcal{K}][\mathcal{S}][\mathcal{V}]$	$\{0, C_{max}\}$	movements at stop $s \in \mathcal{S}$ by vehicle $v \in \mathcal{V}$ at step $k \in \mathcal{K}$
$load$	$[\mathcal{K}][\mathcal{V}]$	$\{0, c_{max}\}$	load of vehicle $v \in \mathcal{V}$ after step $k \in \mathcal{K}$
$time$	$[\mathcal{K}][\mathcal{V}]$	\mathcal{T}	time when vehicle $v \in \mathcal{V}$ arrives at station at step $k \in \mathcal{K}$
$nbBikes$	$[\mathcal{K}][\mathcal{S}]$	$\{0, C_{max}\}$	bikes at stop $s \in \mathcal{S}$ after step $k \in \mathcal{K}$

Table 1. Variables of the step model

In contrast to the routing model, this formulation allows us to directly represent the route of each vehicle by a sequence of stations of fixed length, as shown in Figure 2. This way we can formulate certain constraints more naturally, as we will see in the following description of the model.

Variables. All problem variables are summarized in Table 1: first, we introduce the routing variables $route$, where $route_{k,v}$ denotes the k -th stop in the tour of vehicle $v \in \mathcal{V}$. Thus, the $route$ variables range over the possible stops \mathcal{S}_d . Second, we introduce $service$ variables where $service_{k,s,v}$ represents the number of bikes that are removed or added to station $s \in \mathcal{S}$ at step $k \in \mathcal{K}$ by vehicle $v \in \mathcal{V}$ and therefore ranges over $\{-C_{max}, C_{max}\}$, where $C_{max} = \max_{s \in \mathcal{S}} \{C_s\}$ denotes the maximal capacity over all stations. The load of a vehicle is represented by the $load$ variables where $load_{k,v}$ is the load of vehicle $v \in \mathcal{V}$ at step $k \in \mathcal{K}$. Furthermore, the variables $nbBikes_{k,s}$ state how many bikes are stored at station $s \in \mathcal{S}$ at step $k \in \mathcal{K}$. We introduce $\mathcal{K}_{-1} = \{0, \dots, K-1\}$ for the set of steps excluding the last step and $\mathcal{K}_S = \{1, \dots, K-1\}$ is the set of steps that concern stations, but not the depots (first and last step). we only search on the $route$ and $service$ variables.

$route_{0,v} = D$	$\forall v \in \mathcal{V}$	(3)
$load_{0,v} = \hat{b}_v$	$\forall v \in \mathcal{V}$	(4)
$service_{0,s,v} = 0$	$\forall s \in \mathcal{S}, v \in \mathcal{V}$	(5)
$time_{0,v} = 0$	$\forall v \in \mathcal{V}$	(6)
$nbBikes_{0,s} = b_s$	$\forall s \in \mathcal{S}$	(7)
$activity_{k,s,v} = service_{k,s,v} $	$\forall k \in \mathcal{K}, s \in \mathcal{S}, v \in \mathcal{V}$	(8)
$atleast(activity_{k,v}, 0, S - 1)$	$\forall k \in \mathcal{K}, v \in \mathcal{V}$	(9)
$time_{k,v} \leq time_{k+1,v}$	$\forall k \in \{0, \dots, K - 1\}, v \in \mathcal{V}$	(10)
$service_{k,s,v} \geq 0$	$\forall k \in \mathcal{K}, v \in \mathcal{V}, s \in \mathcal{S} :$	(11)
$service_{k,s,v} \leq 0$	$\forall k \in \mathcal{K}, v \in \mathcal{V}, s \in \mathcal{S} :$	(12)
$load_{k+1,v} = load_{k,v} + \sum_{s \in \mathcal{S}} service_{k+1,v,s}$	$\forall k \in \mathcal{K}_{-1}, v \in \mathcal{V}$	(13)
$nbBikes_{k+1,s} = nbBikes_{k,s} - \sum_{v \in \mathcal{V}} service_{k+1,v,s}$	$\forall k \in \mathcal{K}_{-1}, s \in \mathcal{S}$	(14)
$time_{k+1,v} \geq time_{k,v} + travelTime_{route_{k,v}, route_{k+1,v}}$	$\forall k \in \mathcal{K}_{-1}, v \in \mathcal{V}$	(15)
$(activity_{k,s,v} \geq 0) \Leftrightarrow (route_{k,v} = s)$	$\forall k \in \mathcal{K}, v \in \mathcal{V}, s \in \mathcal{S}$	(16)
$(route_{k,v} = D) \Rightarrow (route_{k+1,v} = D)$	$\forall v \in \mathcal{V}, k \in \{1, \dots, K - 1\}$	(17)
$(route_{k_1,v_1} = route_{k_2,v_2} \wedge route_{k_1,v_1} \neq D) \Rightarrow$	$\forall k_1, k_2 \in \{1, \dots, K - 1\},$	(18)
$time_{k_1,v_1} \neq time_{k_2,v_2}$	$v_1, v_2 \in \mathcal{V}, v_1 \neq v_2$	(18)
$count(route_v, c), \text{ dom}(c, 0, v_{max})$	$\forall v \in \mathcal{V}$	(19)
$load_{K,s,v} = 0$	$\forall s \in \mathcal{S}, v \in \mathcal{V}$	(20)
$route_{K,v} = D$	$\forall v \in \mathcal{V}$	(21)
$service_{K,s,v} = 0$	$\forall s \in \mathcal{S}, v \in \mathcal{V}$	(22)

Table 2. Constraints of the step model

Constraints. All constraints are summarized in Table 2 which we discuss in the following. First, we set up the initial state where step $k = 0$: the first stop of the route of each vehicle v is the depot (Eq. 3) and the initial load of v equals \hat{b}_v (Eq. 4). The initial service is zero (Eq. 5), as well as the initial time (Eq. 6), and the initial number of bikes at station s equals b_s (Eq. 7).

Second, we continue with constraints that render the formulation consistent: first, the activity at station s for vehicle v at step k is always equal to the absolute value of the respective service (Eq. 8). Furthermore, every vehicle $v \in \mathcal{V}$ may only perform actions on at most *one* station at each step k , thus the activity is zero in at least $S - 1$ stations (Eq. 9). Moreover, we state that *time* is always incremental (Eq. 10) and ensure monotonicity (‘sink’ and ‘source’ stations) by stating that those stations that need to receive bikes to reach their target value must have *positive* services (Eq. 11), while stations from which bikes need to be removed to reach their target value, must have *negative* services (Eq. 12).

Third, we state the action constraints that describe how the state changes after each move: first, we update the load of vehicle v after servicing a station at step $k + 1$ (Eq. 13). Then we continue with updating the number of bikes at

station s (Eq. 14) and updating the time at which vehicle v arrives at the station it services at step $k + 1$ (Eq. 15) where $travelTime_{route_{k,v}, route_{k+1,v}}$ is expressed by an `element` constraint.

We link the `route` and `activity` variables (Eq. 16) and state that if vehicle v has returned to the depot before reaching the maximum number of steps, then it may not leave it anymore (Eq. 17). This way we add flexibility with respect to the tour length: vehicles may visit K stations or less. Moreover, we state that two different vehicles cannot visit the same station at the same time (Eq. 18). In Eq. 19 we use a `count` constraint and a temporary variable c to ensure that a station is visited at most v_{max} times in a solution. For the current formulation $v_{max} = 1$, however using a different value enables revisits on the step model.

For the final state, we constrain the load of each vehicle to equal zero (Eq. 20), the K -th stop is the depot (Eq. 21) which has zero service (Eq. 22).

Search Strategy. In our search strategy, we try to construct feasible tours and corresponding loading instructions, for one vehicle after another. Therefore, we search upon the `route` and `activity` variables for each vehicle $v \in \mathcal{V}$: we begin with the `route` in a static order, i.e., $route_{0,v}, \dots, route_{K,v}$, and continue with the loading instructions $activity_{0,v}, \dots, activity_{K,v}$ using a dynamic variable selection, where we select the variable with the largest degree.

In order to obtain a good solution, the value selection for the route variables should return stations that are particularly in need of balancing. Therefore, we have implemented a specialized value selection that returns those stations first that have a particularly high deviation from their target value. For the activity variables, we employ a dynamic max-value selection to achieve a maximal activity at each stop in the route.

Model extensions. Similar to the routing model, the step model can easily be extended to incorporate additional real-world aspects, which are not considered in the current problem statement: waiting times at stations can be included, as well as variable loading times at stations. Furthermore, bike sharing system providers are often interested in a *minimal amount of service* at each station, i.e. a minimal amount of moved bikes per service. This can easily be expressed by applying a lower bound (α) on the `activity` variables.

Other possible extensions are to allow stations as *intermediate* depots by neglecting the ‘sink’ and ‘source’ concept by omitting the respecting constraint, or by allowing loaded vehicles to returned to (or leave) the depot. Finally, we can immediately adapt the model to consider the minimizing the working times in case of full rebalancing by imposing that the final unbalance of the stations should be zero.

4 Large Neighborhood Search

Large Neighborhood Search (LNS) [13] is a local search metaheuristic based on the observation that exploring a *large neighborhood*, i.e., perturbing a signifi-

cant portion of a solution, typically leads to local optima of much *higher quality* than the ones obtained with just slight perturbations. While this is an undoubted advantage in terms of search performance, it does not come without a price as exploring a large neighborhood structure can be *computationally impractical*. For this reason, LNS typically involves *filtering* techniques that allow to keep the neighborhood size under control by removing unfeasible solutions without hindering the search process.

In particular, large neighborhood exploration has been successfully coupled with constraint-based propagation in order to tackle complex routing problems such as VRP with time windows [2, 11].

The customary way of specifying a large neighborhood is to define two steps: (i) a *destroy* step, which takes a solution and relaxes a fraction $d \in [0, 1]$ (the *destruction rate*) of its variables, and (ii) a *repair* step, which takes the relaxed solution and reconstructs a feasible solution by assigning the *free variables*, usually through a greedy heuristic or an exhaustive search, e.g., *Branch & Bound*. Clearly, performing an exhaustive search on a (proper) subset of variables is computationally more tractable than solving the problem in its entirety.

Of course, different values of d originate different neighborhoods and imply different search efforts. For instance, at the most extreme cases, when $d = 1$ the original solution is completely replaced by a new one and local information is lost, while if $d \approx 0$ most of the solution is retained, and only a small neighborhood is explored. By adapting d during the solution process, e.g., based on the search performance, it is possible to codify more sophisticated behaviors, such as *stagnation avoidance*.

Similarly to most metaheuristics also LNS is a template method whose actual implementation depends on problem-specific details. In particular LNS requires to specify the following aspects:

- the way in which the *destroy* step is implemented, i.e., *which variables* are chosen for relaxation;
- the way in which the *repair* step is defined, i.e., random sampling, heuristic search (problem-specific greedy heuristics, ACO, ...) or complete search (depth-first, Branch & Bound, ...);
- whether the search for the *next solution* stops at the first feasible solution, at the first improving solution or continues until a local optimum is found;
- whether d is *evolved* during the search or not and the range of values it can assume;
- whether the *acceptance criterion* is strict improvement, equal quality or it is stochastic, e.g., as in Simulated Annealing;
- the employed *stopping criterion*.

In our approach, most of these aspects are common to both CP models. However, some components, in particular the *destroy* step, are model-specific because they depend on the variables employed for modeling or on the branching strategy. We defer the description of the model-specific components to the last part of this section.

Solution initialization. We obtain the initial solution by performing a tree search with a custom branching strategy tailored for each model. The idea behind our branching strategy is to choose the next station and amount of service so that the total reduction of unbalancing is maximal. Search is stopped after finding the first feasible solution.

Repair step. Similarly to the initialization, the repair step consists of a *Branch & Bound* tree search with a time limit, subject to the constraint that the next solution must be of better quality than the current one. The search starts from the relaxed solution and the time budget is proportional to the number of free variables ($t_{BAB} \cdot n_{free}$) in it. The tree search employs the same branching strategy used for solution initialization.

Acceptance criterion. A repaired solution x_t is accepted as the new best only if it is strictly improving over the previous best x_{best} . If the repair step cannot find an improving solution in the allotted time limit, then *idle iterations* counter ii is increased. When the iteration counter exceeds the maximum number of idle iterations ii_{max} a new initial solution is designated by using a random branching, and the search is restarted.

d update. The destruction rate d evolves during the search in order to implement an intensification/diversification strategy and to avoid stagnation of the search. In our implementation at each step its value is updated as follows:

$$d = \begin{cases} \min(d \cdot 1.05, 0.8) & \text{if } x_t > x_{best} \\ d = d_{init} & \text{otherwise} \end{cases} \quad (23)$$

By using this update scheme, when the repair step cannot find an improving solution in a given neighborhood, then the *radius* of the neighborhood is increased so to allow solution diversification with the aim of exploring different regions of the search space.

As soon as a new best solution is found, the original initial neighborhood radius is reset, so that the exploration of the newly discovered solution region is intensified.

Stopping criterion. We allow the algorithm to run for a given timeout, when the time is up, the algorithm is interrupted and the best solution found is returned.

Destroy step. As mentioned before, the only model-specific component of our implementation is the destroy step. In fact, this is the most relevant aspect of LNS since it requires a careful selection of the variables that have to be relaxed to define the neighborhood. This selection strongly depends on some specific knowledge about the problem structure in order to avoid unmeaningful combinations.

Destroy step for the Routing model. In the case of the routing model, the relaxed solution is generated by selecting $d \cdot |R_i|$ stations from each route R_i and resetting the *succ*, *service*, and *vehicle* variables of these stations to their original domains. Moreover, also the *succ* variable of the stations preceding the relaxed ones are reset to their original domain to allow for different routes. Note that since we are considering also these variables the final fraction of variables relaxed is in fact greater than d .

Destroy step for the Step model. The relaxed solution of the step model is produced by selecting $d \cdot \sum_i |R_i|$ internal nodes (i.e., excluding the depots) from all the routes and resetting the *route* and *service* variables.

5 Computational Evaluation

In this section we report and discuss the experimental analysis of the algorithms. All the experiments were executed on an Ubuntu Linux 12.04 machine with 16 Intel[®] Xeon[®] CPU E5-2660 (2.20GHz) cores. For fair comparison, both the CP and the LNS algorithms were implemented in Gecode (v 3.7.3) [7], the LNS variant consisting of a specialized search engine and two specialized branchers.

The LNS parameters (i_{max} , d_{init} and t_{BAB}) have been tuned by running an *F-Race* [3] with a confidence level of 0.95 over a pool of 150 benchmark instances from Citybike Vienna. Each instance, featuring a given number of stations $S \in \{10, 20, 30, 60, 90\}$, was considered with different number of vehicles $V \in \{1, 2, 3, 5\}$ and time budgets $t \in \{120, 240, 480\}$, totaling 900 problems. The tuning was performed by letting the algorithms run for 10 minutes. The best configurations were $i_{max} = 40$ and $t_{BAB} = 400$ for both models, $d_{init} = 0.05$ for the routing model and $d_{init} = 0.1$ for the step model. Note that the way the *destroy* step is designed determines the optimal value of d_{init} , as a consequence in both models a similar proportion of variables is relaxed (about 10 – 20%).

For benchmarking, we let the winning configurations for LNS and the pure CP models run for one hour, the results are summarized in Table 3.

5.1 Model and Solution Method comparison

The main goal of this comparison is to understand and analyze the behavior of the CP Branch & Bound and LNS solution methods for the two problem models. Figure 3 shows exemplarily the evolution of the best cost within one search run on an instance from the Citybike Vienna benchmark set featuring 30 stations. The pink and turquoise dashed lines represent the resolution using branch and bound respectively on the routing and the step model. The solid lines represent the median of 10 runs of LNS on the two models. The dark areas represent the interquartile range at each time step, while the light areas represent the maximum range covered by LNS over the 10 runs.

From the plot it is possible to see that, regarding the pure CP approaches (i.e., Branch & Bound), the routing model is clearly outperforming the step model. As

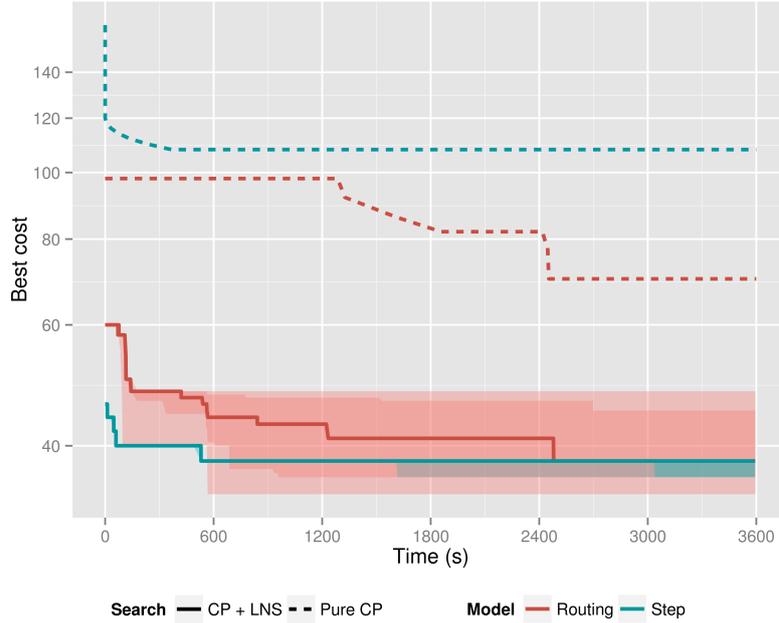


Fig. 3. Evolution of the best cost for the Pure CP and LNS solution methods for the routing and the step model (30 stations, 2 vehicles, time budget 480 minutes)

for the LNS-based solvers, the situation is quite the opposite, with the step model outperforming the routing model on the median run. One must however consider that performance data collected on a single instance is of limited statistical significance. As for the comparison between pure CP approaches and LNS-based ones, the latter exhibit better *anytime* performance, reaching low areas of the objective function much faster than their Branch & Bound counterparts. Of course this comes at the price of *completeness*, and we expect CP approaches to rival with or outperform the LNS-based ones given enough time. It is worth noticing that this result is consistent across the whole benchmark set.

5.2 Comparison with other methods

In this second experiment, we compare our CP and LNS solution methods with the state-of-the-art results of [9], who solved the same set of instances using a Mixed Integer Linear Programming solver (MILP) and a Variable Neighborhood Search (VNS) strategy. The result of the comparison against the best of the three different VNS approaches in [9] are reported in Table 3. The reported results in each row are averages over 150 instances, grouped by size, number of vehicles and available time for the trucks to complete the tour.

Cells marked with a dash refer to instance classes for which the algorithm cannot reach a feasible solution within a hour. In these cases it makes no sense

to compute a mean, thus for the CP models we report the number of instances in which the cost is inferior, equal or superior to the one obtained by the MILP solver in [9]. Of course, this may also happen in the case of LNS, since the initial solution is obtained through a Branch & Bound search.

In this table we did not report the results of our former ACO+CP solution approach [6], since it was outperformed by the methods proposed in this paper.

From the table it is possible to observe that the VNS heuristics proposed in [9] consistently outperform our pure CP and LNS-based solution methods on all the instance classes. On the other hand our LNS approach based on the routing model, has very close performances to [9] on almost all instances with 30 stations or less, although requiring more time to reach the same result.

As for the comparison with MILP, our CP models solved by Branch & Bound are able to match or outperform the upper bound solution found by MILP on the mid- and big-size instances ($S \geq 30$). Moreover, the routing model and the step model seems to have complementary strengths and weaknesses on the whole benchmark, with the step model being able to consistently find solutions on instances that are hard for the routing model and the other way round.

Overall, our LNS approach appears more robust with respect to the largest instances, where pure CP often fails to find even a feasible solution. However, similarly to the Branch & Bound solution method, also in this case there is no clear winner.

6 Conclusions

In this paper, we have presented two novel CP models for the problem of balancing bike sharing systems (BBSS), and a Large Neighbourhood Search (LNS) approach based on the propagation of the constraints defined in each model for obtaining good solutions in a reasonable time.

We have compared the results of our research against the state-of-the-art VNS and MILP solvers for BBSS proposed in [9]. Even though our approaches are not able to outperform the current bests, our results are reasonably close, and the two models we propose are based on a more general formulation of the BBSS problem that, for example, involves the possibility of visiting the same station repeatedly over the tour or to take into consideration the loading times.

Furthermore, we experimentally show that combining the power of constraint propagation with neighborhood search is a natural and effective way to trade *completeness* for *performance*. In fact, the LNS approaches based on our two CP models, consistently outperform their Branch & Bound counterparts by exploiting constraint propagation to limit the size of the neighborhood and reaching low-cost solutions very quickly.

As future work, we plan to consider different variants of LNS, e.g., employing different stopping conditions and acceptance criterions. Moreover, we are interested in solving the *dynamic* variant of the BBSS problem, where bikes are moved independently from station to station during the rebalancing, resulting in variable target values and variable station loads over time.

Instance		CP		CP / MILP		MILP [9]		LNS		VNS [9]				
S	V	Routing	Step	Routing	Step	\bar{u}	\bar{t}	Routing	Step	\bar{f}	\bar{t}			
	\hat{t}	\bar{f}	\bar{t}	$</= / >$	$</= / >$			\bar{f}	\bar{t}					
10	1	120	28.3348	149	35.2678	0	2/28/0	1/8/21	28.3348	53	35.4012	0	28.3348	2
10	1	240	4.2694	398	17.5353	334	2/28/0	0/1/29	4.2694	148	17.5353	240	4.2694	10
10	1	480	0.0032	320	14.5356	462	9/21/0	0/0/30	0.0033	128	13.1356	180	0.0032	17
10	2	120	10.4693	445	18.8022	52	0/22/8	0/4/26	9.8027	138	18.8689	23	9.9360	3
10	2	240	0.0034	56	13.6025	459	5/25/0	0/0/30	0.0034	65	13.1358	269	0.0034	19
10	2	480	0.0032	101	14.0691	644	2/28/0	0/0/30	0.0033	101	13.1358	460	0.0032	15
20	2	120	63.4030	1042	72.1357	260	1/3/26	0/1/29	55.8029	251	72.4024	131	55.3363	8
20	2	240	23.1388	1083	30.2712	389	11/0/19	4/0/26	19.7388	292	27.5378	668	4.2058	58
20	2	480	14.6057	1089	20.0054	635	4/0/26	1/0/29	1.8091	306	17.7386	721	0.0061	142
20	3	120	48.2043	1391	54.3369	310	3/0/27	1/1/28	37.3376	250	52.2035	280	31.7376	13
20	3	240	20.3392	1347	20.6056	616.3439	4/0/26	5/0/25	6.1408	285	17.1389	742.5992	0.0065	65
20	3	480	14.6724	1503	—	—	8/0/22	4/0/26	13.3419	311	—	—	0.0061	114
30	2	120	114.8696	475	122.4692	250	2/3/25	1/2/27	106.9363	264	123.4691	68	104.7363	12
30	2	240	59.3392	936	71.6047	439	23/1/6	14/0/16	74.9389	256	69.7379	626	34.6061	109
30	2	480	22.9425	1320	22.4748	558.2628	24/0/6	25/0/5	69.7407	338	20.0080	938.3446	0.0093	491
30	3	120	95.2044	612	104.6036	440	6/0/24	4/0/26	90.4042	240	102.5369	265	78.1377	21
30	3	240	36.1417	1151	40.6069	645	26/0/4	24/0/6	61.6072	272	39.4068	450.9957	7.0752	191
30	3	480	22.9425	1333	—	—	30/0/0	28/0/2	175.4000	341	—	—	0.0093	399
60	3	120	284.0045	1311	287.7370	265.5874	5/0/25	7/0/23	274.2710	228	282.5379	228	253.8046	45
60	3	240	—	—	185.5412	146.4551	23/0/7	30/0/0	370.2000	270	184.3425	270	126.7428	521
60	3	480	67.2173	616	76.0806	541.0620	30/0/0	30/0/0	—	205	74.3471	625.0877	6.6176	3600
60	5	120	242.2741	1427	—	—	24/0/6	22/0/8	289.2711	268	242.4741	268	196.6075	99
60	5	240	115.8813	1323	107.3459	287.2886	30/0/0	30/0/0	370.2000	273	110.8813	273	41.4816	1556
60	5	480	55.5532	1259	—	—	30/0/0	30/0/0	—	259	35.0875	—	0.0190	3600
90	3	120	480.3379	1761	—	—	13/1/16	14/0/16	492.2032	335	476.4046	335	441.6047	82
90	3	240	—	—	364.4748	339.3373	0/0/30	30/0/0	566.2667	352	370.1427	352	294.4765	985
90	3	480	—	—	217.2814	561.5618	0/30/0	30/0/0	—	367	216.2811	548.0430	100.9522	3600
90	5	120	436.8076	2071	—	—	30/0/0	29/0/1	566.2667	429	438.0076	429	376.0743	169
90	5	240	—	—	262.7465	511.1937	0/30/0	30/0/0	—	415	262.4129	844.3994	174.2157	3304
90	5	480	87.6287	752	88.7570	354.9604	30/0/0	30/0/0	—	350	85.3567	667.9110	1.4285	3600

Table 3. Comparison of our approaches with the MILP and the best VNS approach of [9].

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. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science* 38(4), 515–530 (2004)
3. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. *Experimental methods for the analysis of optimization algorithms* pp. 311–336 (2010)
4. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. To appear in *Discrete Optimization* (2012)
5. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Tech. Rep. CIRRELT-2012-09, CIRRELT, Montreal, Canada (2012), submitted to *Transportation Science*
6. Di Gaspero, L., Rendl, A., Urli, T.: A hybrid ACO+CP for balancing bicycle sharing systems. In: Blesa Aguilera, M.J., Blum, C., Festa, P., Roli, A., Sampels, M. (eds.), *Hybrid Metaheuristics 8th International Workshop, HM 2013 Ischia, Italy, May 23-25, 2013 Proceedings*, volume 7919 of *Lecture Notes in Computer Science*, Springer (2013), an abridged version is available from <http://satt.diegm.uniud.it/uploads/Papers/DiRU13a.pdf>
7. Gecode Team: Gecode: Generic constraint development environment (2006), available from <http://www.gecode.org>
8. Kilby, P., Shaw, P.: Vehicle routing. In: Rossi, F., Beek, P.v., Walsh, T. (eds.) *Handbook of Constraint Programming*, chap. 23, pp. 799–834. Elsevier Science Inc., New York, NY, USA (2006)
9. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In: Middendorf, M., Blum, C. (eds.) *Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science*, vol. 7832, pp. 121–132. Springer (2013)
10. Raviv, T., Tzur, M., Forma, I.A.: Static Repositioning in a Bike-Sharing System: Models and Solution Approaches. To appear in *EURO Journal on Transportation and Logistics* (2012)
11. Rousseau, L.M., Gendreau, M., Pesant, G.: Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics* 8(1), 43–58 (2002), <http://dx.doi.org/10.1023/A:1013661617536>
12. Schuijbroek, J., Hampshire, R., van Hoesve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Tech. Rep. 2013-E1, Tepper School of Business, Carnegie Mellon University (2013), http://www.contrib.andrew.cmu.edu/~vanhoeve/papers/bike_sharing.pdf
13. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.F. (eds.) *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings. Lecture Notes in Computer Science*, vol. 1520, pp. 417–431. Springer (1998)