A Hybrid ACO+CP for Balancing Bicycle Sharing Systems

Luca Di Gaspero[1], Andrea Rendl[2], and Tommaso Urli[1]

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

[2] DTS Mobility Department
Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
andrea.rendl@ait.ac.at

Abstract. Balancing bike sharing systems is an increasingly important problem, because of the rising popularity of this mean of transportation. Bike sharing systems need to be balanced so that bikes (and empty slots for returning bikes) are available to the customers, thus ensuring an adequate level of service.

In this paper, we tackle the problem of balancing a real-world bike sharing system (BBSS) by means of a hybrid metaheuristic method. Our main contributions are: *(i)* a new Constraint Programming (CP) formulation for the problem, and *(ii)* a novel hybrid approach which combines CP techniques with Ant Colony Optimization (ACO). We then validate our approach against real world instances from the Vienna Citybike system.

1 Introduction

The idea of bike sharing is to provide bikes to the citizens via stations that are located all around the city. At each station, bikes are stored in special racks, such that users can easily pick up or return a bike. However, popular stations are often emptied or filled very quickly, resulting in annoyed users who cannot return or retrieve bikes. To avoid this, the stations must be balanced.

Bike sharing systems are balanced by distributing bikes from one station to another by using specific vehicles. Therefore, balancing the system corresponds to finding a tour for each vehicle, including loading and unloading instructions per station such that the resulting system is balanced. Clearly, balancing bike sharing systems is a difficult task, since it requires solving a vehicle routing problem combined with distributing single commodities (bikes) according to the target values at the stations.

In the following, we are consistent with the notation introduced in [1]. We consider balancing a bike sharing system with S stations $\mathcal{S} = \{1, \ldots, S\}$ and a set of depots $\mathcal{D} = \{S+1, \ldots, S+D\}$, where each station $s \in \mathcal{S}$ has a capacity

$C_s > 0$, a number of available bikes b_s and the number of target bikes t_s that denotes the number of bikes that should be at station s after balancing the system. We use V vehicles $\mathcal{V} = \{1, \ldots, V\}$ with capacity $c_v > 0$ and initial load $\hat{b}_v \geq 0$ that distribute the bikes within maximal $\hat{t}_v > 0$ time units. The travel times between stations (and the depots) is given by a travel time matrix $tt_{u,v}$ where $u, v \in \mathcal{S} \cup \mathcal{D}$, which includes also an estimate of the processing times needed to serve the station, if $v \in \mathcal{S}$.

We want to achieve a maximally balanced system where each vehicle travels on a minimal route. Therefore, in our cost function, we minimize the sum of the deviations from the target value for each station and include both the travel distance and the overall activity of each vehicle as a measure of the work effort.

In this paper, we first introduce a novel Constraint Programming (CP) model that is based on a vehicle routing formulation. Then we show how we can generally combine CP with Ant Colony Optimization (ACO) by utilizing ACO as search engine in the CP solving process. The combination is based on the idea of tackling the problem as a bi-level optimization problem, in which the routing variables are handled by ACO, whereas the operation variables (which model the number of bikes to load or unload) are taken care of by CP. Finally, we show with an experimental evaluation on real-world instances from Citybike Vienna, that the hybrid ACO+CP approach outperforms the pure CP formulation.

2 Related work

A few approaches for integrating ACO and CP are available from the literature. The first attempt is due to Meyer and Ernst [2], who apply the method for solving a Job-Shop Scheduling problem. The proposed procedure employs ACO to learn the learning strategy used by CP in the tree-search. The solutions found by CP are fed back to ACO, in order to update its probabilistic model. In this approach, ACO can be conceived as a master online-learning branching heuristic aimed at enhancing the performance of a slave CP solver.

A slightly different approach has been taken by Khichane et al. [3, 4]. Their algorithm works in two phases. At first CP is employed to sample the space of feasible solutions and the information collected is processed by the ACO procedure for updating the pheromone trails. In the second phase, the pheromone information is employed as the value ordering used for CP branching. Unlike the previous one, this approach uses the learning capabilities of ACO in an offline fashion. More standard approaches in which CP is used to keep track of the feasibility of the solution constructed by ACO and to reduce the domains through Constraint Propagation have been used by a number of authors. This idea has been applied to Job-Shop Scheduling [2] and Car Sequencing [5].

Our approach also shares some similarities with Large Neighborhood Search (LNS) [6] in that *(i)* we exploit constraint propagation to reduce the domains of the variables and *(ii)* we handle different subsets of variables separately. However, unlike LNS, our search process includes a learning component. Moreover, our separate treatment of variables is motivated by the good performance of ACO on routing problems, rather than by a need for a better neighborhood exploration.

Balancing of bike sharing systems has become an increasingly studied problem in the last few years. Benchimol et al. [7] 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. Contardo et al. [8] 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. [9] 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. [10], 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. [1] 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. [11] 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 an scheduling formulation of the problem and therefore differs significantly from our VRP-based formulation.

3 A Constraint Model for BBSS

Our constraint model is based on the constraint model [12] of the classical Vehicle Routing Problem (VRP) that is concerned with servicing a set of customers with a fleet of vehicles with cost-optimal tours. The VRP model employs successor and precessor variables to represent the path of each vehicle on a special graph G_{VRP} that consists of three different kinds of nodes: first, the starting node for each vehicle (typically the respective depot), second, the nodes that should be visited in the tour, and third, the end nodes for each vehicle, again typically the respective depot. In summary, G_{VRP} contains $2V + S$ nodes, where V is the number of vehicles and S is the number of nodes to visit. This graph structure allows to easily define successor and predecessor variables to represent paths.

We extend the VRP model to allow unvisited stations and to capture loading instructions on a per-station basis. To achieve this we introduce a dummy vehicle that (virtually) visits all the unserviced stations. This formulation allows to treat unvisited stations as a cost component, and makes it easier to ensure that no operations are scheduled for unvisited stations by constraining the load of the dummy vehicle to be always zero. This results in an extension of G_{VRP} to graph G_{BBSP} that contains $2(V + 1) + S$ nodes, where $V + 1$ is the number of vehicles

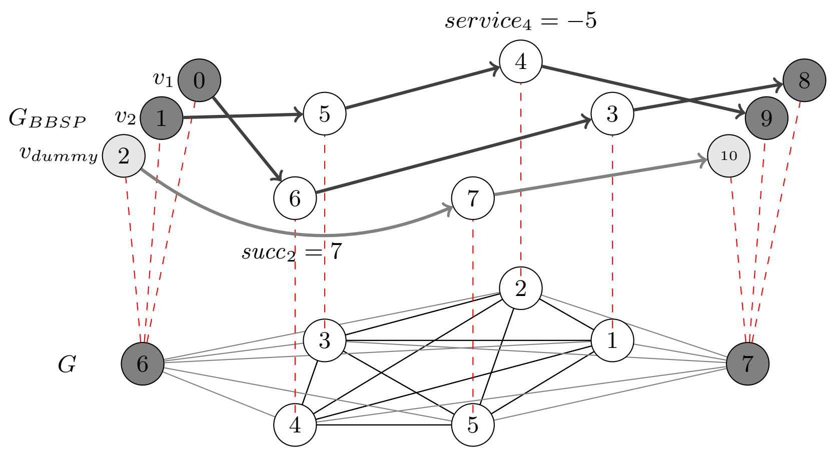


Fig. 1. Graph encoding of the BBSS problem employed in the routing CP model. The lower layer shows the original graph, whereas the upper layer shows the encoded graph in the case of two vehicles, and a solution. The path starting at node 2 and ending at node 10 (i.e., the dummy vehicle) corresponds to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

including the dummy vehicle. This encoding is illustrated in Figure 1, where the basic structure is shown on the lower layer, and the encoded G_{BBSS} and a possible solution is shown on the upper layer. We represent all nodes in G_{BBSP} in the ordered set $\mathcal{U}$ which is defined as follows:

$$
\begin{array}{rl|l}
\mathcal{U} = \{ & 0, \dots, V, & \mathcal{V}_s\text{: start nodes} \\
& V+1, & \text{station 1} \\
& V+2, & \text{station 2} \\
& \dots, & \dots \\
& V+S, & \text{station } S \\
& V+S+1, \dots, 2V+S+2 \; \} & \mathcal{V}_e\text{: end nodes}
\end{array}
$$

Thus, $\mathcal{U}$ first contains the starting nodes (depots) for the V vehicles and the dummy vehicle, followed by the S regular stations, and finally the end nodes (depots) for V vehicles and the dummy. Note, that we denote $\mathcal{V}_s = \{0, \dots, V\}$ the set of start nodes of vehicles and $\mathcal{V}_e = \{V+S+1, \dots, 2V+S+2\}$ the set of end nodes of each vehicle. Thus, $\mathcal{U} = \mathcal{V}_s \cup \mathcal{S} \cup \mathcal{V}_e$. In summary, the tour of vehicle $v \in \mathcal{V}$ starts at a depot in $\mathcal{V}_s$, continues to some station nodes in $\mathcal{S}$ and ends at a depot in $\mathcal{V}_e$. In the following, we give a detailed description of our model.

3.1 Variables

The first set of variables are the successor variables that represent the paths by defining the successor of each node in $\mathcal{U}$. Thus, we have $|\mathcal{U}|$ successor variables *succ* that range over $\mathcal{U}$, where $succ_i$ represents the node following node i. In

Table 1. Variables in the CP Model

name[dimension]	domain	description
succ $[\mathcal{U}]$	$\mathcal{U}$	successor of node $i \in \mathcal{U}$
pred $[\mathcal{U}]$	$\mathcal{U}$	predecessor of node $i \in \mathcal{U}$
vehicle $[\mathcal{U}]$	$\mathcal{U}$	vehicle serving node $i \in \mathcal{U}$
service $[\mathcal{U}]$	$[\pm max(C_{max}, c_{max})]$	removed/added bikes at node $i \in \mathcal{U}$
load $[\mathcal{U}]$	$[0, c_v]$	load of vehicle v after serving node $i \in \mathcal{U}$
time $[\mathcal{U}]$	$[0, \hat{t}_v]$	time when vehicle v arrives at node $i \in \mathcal{U}$
loadTime $[\mathcal{U}]$	$[0, \hat{L}]$	loading time at node $i \in \mathcal{U}$
deviation $[\mathcal{S}]$	$\mathcal{S}$	deviation from target at station $s \in \mathcal{S}$
cost	$[l, u]$	overall cost of the solution

addition, we define predecessor variables *pred* where $pred_i$ denotes the node which comes just before node i in the route. Though redundant, predecessor variables channelled with successor variables result in stronger propagation [12]. Second, we associate a vehicle to each node i by the variable $vehicle_i$ that ranges over $\{0, \ldots, V\}$. The loading instructions for each node are captured by operation variables *service* where $service_i$ represents the number of bikes that are added or removed at node $i \in \mathcal{U}$ and ranges over $[\pm max(C_{max}, c_{max})]$ where C_{max} and c_{max} are respectively the maximum capacities of stations and vehicles. We also introduce load variables $load_i$, which represent the load of the vehicle after visiting node $i \in \mathcal{U}$. Next come the time-related variables: $time_i$ constitutes the arrival time at which a vehicle arrives at node i. In our problem formulation, the arrival time also includes the processing time, i.e., the time for loading/unloading the vehicle at that node. Finally, we use S deviation variables *deviation* where $deviation_s$ represents the deviation from the target values at station $s \in \mathcal{S}$ after the balancing tours. Variables are summarized in Tab. 1.

3.2 Constraints

We divide the introduction of the constraints in the model by first stating the essential constraints that are required to comprehensively model the problem, and then discussing some redundant constraints that will help the solution process.

Essential constraints. We start our description with the routing constraints for the path: all successors and predecessors take different values.

$$\texttt{alldifferent}(succ) \tag{1}$$

$$\texttt{alldifferent}(pred) \tag{2}$$

Note that, while these constraints are alone not sufficient to eliminate subtours from the solutions, according to [12] the presence of a finite time horizon for vehicles (which is the case for BBSS) ensures the absence of cycles. In the case of the dummy vehicle, which has no finite horizon, a similar task is carried out by a symmetry breaking constraint which enforces an ordering of the nodes in the route, effectively making subtours impossible to occur.

Then we set the successor-predecessor chain for each regular station

$$pred_{succ_s} = s \quad \forall s \in \mathcal{S} \tag{3}$$
$$succ_{pred_s} = s \quad \forall s \in \mathcal{S} \tag{4}$$

and the successor-predecessor chain for the start and end nodes where $\hat{s} = V+S$ represents the index of first end node in $\mathcal{U}$:

$$pred_v = \hat{s} + v \quad \forall v \in \mathcal{V}_s \tag{5}$$
$$succ_{\hat{s}+v} = v \quad \forall v \in \mathcal{V}_s \tag{6}$$

Furthermore, no loops are allowed in the paths, i.e.

$$pred_i \neq i \quad \forall i \in \mathcal{U} \tag{7}$$
$$succ_i \neq i \quad \forall i \in \mathcal{U} \tag{8}$$

We continue with constraints on the vehicle variables. First, we set the respective vehicle $v \in \mathcal{V}_s$ for each start- and end-node in the path:

$$vehicle_v = v \quad \forall v \in \mathcal{V}_s \tag{9}$$
$$vehicle_{\hat{s}+v} = v \quad \forall v \in \mathcal{V}_s \tag{10}$$

and second, we set the vehicle-chain over the path variables:

$$vehicle_{succ_i} = vehicle_i \quad \forall i \in \mathcal{U} \tag{11}$$
$$vehicle_{pred_i} = vehicle_i \quad \forall i \in \mathcal{U} \tag{12}$$

For the loading constraints, we first set the initial load $\hat{b}_v$ and constrain the dummy vehicle to be empty

$$load_v = \hat{b}_v \quad \forall v \in \mathcal{V}_s \setminus \{V\} \tag{13}$$
$$load_V = 0 \tag{14}$$

and continue with the loading restrictions along a path

$$load_{succ_i} = load_i - service_i \quad \forall i \in \mathcal{U} \tag{15}$$

Finally, every vehicle must be completely empty at the end of the route, i.e.:

$$load_v = 0 \quad \forall v \in \mathcal{V}_e \tag{16}$$

Additionally, we constrain the load for vehicles: if station s is not served by the dummy vehicle (V), then the service must not be zero, and vice versa:

$$(vehicle_s \neq V) \iff (service_s \neq 0) \quad \forall s \in \mathcal{S} \tag{17}$$

Furthermore, the load of the vehicle after visiting station $s \in \mathcal{S}$ may not exceed its capacity c:

$$load_s \leq c_{vehicle_s} \quad \forall s \in \mathcal{S} \tag{18}$$

Next come the operation constraints. At first we impose *operation monotonicity*, i.e., services at station s should either force loading or unloading bikes depending on the current number of bikes b_s and the target value of bikes t_s:

$$service_s \leq 0 \quad \forall_{s \in \mathcal{S}} : b_s > t_s \tag{19}$$
$$service_s \geq 0 \quad \forall_{s \in \mathcal{S}} : b_s < t_s \tag{20}$$

Notice that a service value of 0 is admissible in both cases since a station could remain unserved (e.g., because of the time budget constraints). The service at the start and end nodes (depots) i is zero for all vehicles:

$$service_i = 0 \quad \forall i \in \mathcal{V}_s \tag{21}$$
$$service_i = 0 \quad \forall i \in \mathcal{V}_e \tag{22}$$

Furthermore, the service is limited by the maximal number of bikes in the station, and we cannot have a negative number of bikes:

$$b_s + service_s \leq C_s \quad \forall s \in \mathcal{S} \tag{23}$$
$$b_s + service_s \geq 0 \quad \forall s \in \mathcal{S} \tag{24}$$

Finally, we state the time constraints, where we begin with setting the arrival time (and processing time) at the start depots to zero

$$time_v = 0 \quad \forall v \in \mathcal{V}_s \tag{25}$$

and set the time chain for the successor and predecessor variables:

$$time_v = time_{pred_v} + tt_{pred_v,v} \quad \forall v \in \mathcal{S} \cup \mathcal{V}_e \tag{26}$$
$$time_{succ_v} = time_v + tt_{v,succ_v} \quad \forall v \in \mathcal{V}_s \cup \mathcal{S} \tag{27}$$

At last, the overall working time for each vehicle must be within its time budget:

$$time_{\hat{s}+v} \leq \hat{t}_v \quad \forall v \in \mathcal{V} \tag{28}$$

This concludes the description of the essential of our CP model for the BBSS problem. The model can be enhanced by some redundant constraints, that will take care of some particular substructure of the problem.

Redundant constraints. First, because of the monotonicity constraints (19–20), the stations requiring the unloading of bikes must be removed from the successors of the starting depots

$$succ_i \neq j \quad \forall i \in \mathcal{V}_s, j \in \{s \in \mathcal{S} | b_s < t_s\} \tag{29}$$

Similarly, because of constraint (16), which requires empty vehicles at the end of the path, the stations requiring the loading of bikes must be removed from the predecessors of the ending depots

$$pred_i \neq j \quad \forall i \in \mathcal{V}_e, j \in \{s \in \mathcal{S} | b_s > t_s\} \tag{30}$$

Finally, an early failure detection of the working time constraint (28) is possible. If the working time of the current partial solution plus the time to reach the final depot exceeds the total time budget, then the solution can't be feasible.

$$time_i + tt_{i,\hat{s}+vehicle_i} \leq \hat{t}_{vehicle_i} \quad \forall i \in \mathcal{S} \tag{31}$$

Cost function. The cost function of the problem is a hierarchical one, and comprises two different major components: the level of unbalancing and the working effort.

The unbalancing component is defined in terms of the *deviation* variables, which are set to be the absolute value of the deviation from the target number of bikes at each station after service has been performed, i.e.:

$$deviation_s = |b_s + service_s - t_s| \quad \forall s \in \mathcal{S} \tag{32}$$

The working effort is the sum of the total traveling time (i.e., the sum of the times at which each vehicle reaches its ending depot) plus the overall activity performed throughout the path (i.e., the absolute value of the *service*).

The cost function is the weighted aggregation of the two components, i.e.:

$$cost = w_1 \sum_{s \in \mathcal{S}} deviation_s + w_2 (\sum_{v \in \mathcal{V}} time_{\hat{s}+v} + \sum_{s \in \mathcal{S}} |service_s|) \tag{33}$$

where $w_1 = 1$ and $w_2 = 10^{-5}$, so that the satisfaction of the first component prevails over the second one. This cost function, defined in [1], is the *scalarization* of a multi-objective problem in nature, thus some points in the Pareto optimal set are neglected by construction. The main reason for this choice was the need to compare with the current bests, moreover, to the best of our knowledge, research in multi-objective propagation techniques is still at an early stage.

4 An ACO+CP Hybrid

Ant Colony Optimization [13] is an iterative constructive metaheuristic, inspired by the ant foraging behavior. The ACO construction process is driven by a probabilistic model, based on pheromone trails, which are dynamically adjusted by a learning mechanism. Constraint Programming(CP) [14] is an exact solving approach where a constraint model is solved using a customized search strategy interleaved with strong filtering (propagation) of the variables' domains.

The hybridization of ACO and CP is described in Algorithm 1 and is, in its essence, a bi-level optimization process. The basic idea is to partition the set

Algorithm 1: ACO + CP

```
   input : $X = X_{Ants} \cup X_{CP}$, a set of integer variables partitioned into variables
           dealt with ACO and CP, respectively
           $\mathcal{C}$, a set of constraints
           $\mathcal{F}$, a cost function
1  initialize all pheromone trails to $\tau_{\text{start}}$;
2  $g \leftarrow 0$;
3  repeat
4      for $k \in \{1, \ldots, n\}$ do
5          $\mathcal{A}_k \leftarrow \emptyset$;
6          repeat
7              select a variable $x_i \in X_{Ants}$, so that $x_i \notin \text{var}(\mathcal{A}_k)$, and a value
               $j \in D_j$ according to the pheromone trail $\tau_{ij}$ (and possibly the
               heuristic information $\eta_{ij}$);
8              add $\{x_i := j\}$ to $\mathcal{A}_k$;
9              if Propagate($\mathcal{A}_k, \mathcal{C}$) = $Failure$ then
10                 Backtrack($\mathcal{A}_k$);
11         until var($\mathcal{A}_k$) = $X_{Ants}$;
12         TreeSearch($X, \mathcal{C}, \mathcal{F}$);
13         update pheromone trails using $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ and $\mathcal{F}$;
14         $g \leftarrow g + 1$;
15 until TerminateSearch($g, \mathcal{A}_i$, time);
```

X of problem variables into two sets X_{Ants} and X_{CP}. The values for the X_{Ants} variables are dealt with by an ACO procedure and once they are set, a tree-search (line 12) finds the values for the remaining variables. The tree-search procedure can be either a branch-and-bound algorithm (exploiting the information of the cost function $\mathcal{F}$) or a (possibly) faster depth-first-search if we are satisfied with a good assignment of the X_{CP} variables.

Once all n ants have found a solution, the pheromone trails are updated according to the solution components and their cost value. The overall search is typically stopped at a given timeout. It is worth noticing that, in the proposed approach, the interaction between ACO and CP is two-way. The pruning capabilities of constraint propagation are employed both to restrict the number of alternatives the ants must face at each choice point (see line 7), and as a mechanism for early failure detection (see line 9). On the other hand, ACO helps CP converging faster, by avoiding search paths that lead to unfeasible solutions.

4.1 ACO+CP for BBSS

In our CP model for the BBSS problem, there is a natural partition of the decision variables into two families, i.e., routing and operation variables.

Handling of routing variables. The first set of variables, $succ_i$, is handled very naturally by ACO, which has been shown to be particularly effective in solving routing problems. In our approach, ACO is embodied by a two-phase branching strategy which takes care both of variable and value selection. This process is illustrated in Figure 4.1.

Variable selection. The first variable to be selected, according to the heuristic, is the *succ* of the first vehicle starting depot (Fig. 2(a)). As for the next variable to assign, we always choose the one indicated by the value of the last assigned variable, i.e., the *succ* of the last assigned node (Fig. 2(b)). By following this heuristic, we enforce the completion of existing paths first. If the successor of the last assigned node is a final depot (Fig. 2(c)), then we cannot proceed further on the current path, and we start a new one by assigning the successor of the next starting depot. Once the paths of all vehicles are set, the remaining unserved nodes will be assigned to the dummy vehicle (Fig. 2(d)).

Value selection. Once the next variable to assign is chosen, all the values in its current domain are considered as candidates. Note that, in this, we are in fact exploiting problem-specific knowledge, as the domain of a variable is, at any time, determined by the constraint propagations activated earlier in the search.

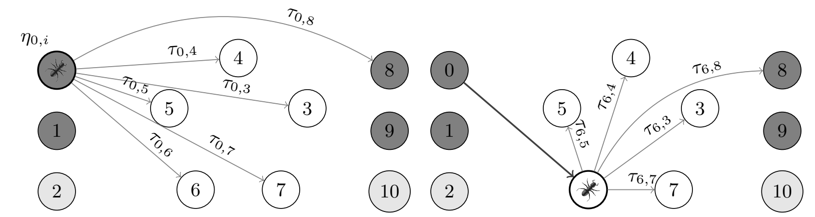
The next step is where ACO comes into play. For our approach we have chosen a popular ACO variant known as the hyper-cube framework for ACO (HC-ACO) [15]. As most other ACO approaches, HC-ACO maintains a pheromone table in which each $\langle X_i, v_j \rangle$ (variable, value) pair has a corresponding $\tau_{i,j}$ pheromone value indicating the desirability of value v_j for the variable X_i. The advantage of HC-ACO over other ACO variants, is that the update rule for pheromones involves a normalization factor which makes the approach independent of the scale of the cost function and doesn't require to enforce a $[\tau_{min}, \tau_{max}]$ interval.

In line with the majority of ACO variants, our value selection heuristic is stochastic, with the probability of choosing a specific value being proportional to the corresponding τ-value. In particular, the probability $P(X_i, v_j)$ of choosing the value v_j for the variable X_i is

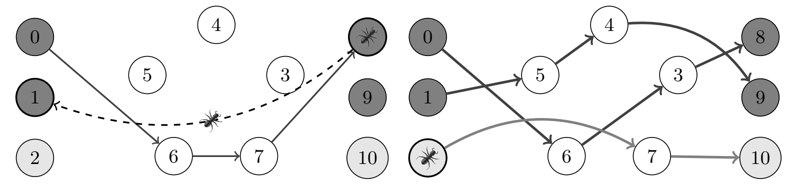
$$P(X_i, v_j) = \frac{\tau_{i,j}}{\sum_{v_k \in dom(X_i)} \tau_{i,k}} \tag{34}$$

Handling of operation variables. The operation variables are assigned through depth-first tree-search, based on *deviation* variables, which are the main component of our cost function. This way, employing a min value heuristic, lower cost solutions are produced before bad ones.

While other choices are possible, e.g. a full exploration of the tree by branch-and-bound, in this context we aim at finding quickly feasible solutions, so that they can be used for learning. The rationale behind this choice is that decisions taken towards the root of the search tree have a greater impact than the ones taken towards the leaves, and τ-updates are the only way to improve our ACO-based value selection heuristic.



(a) The ant is first placed at the starting depot of the first vehicle. (b) The value of the $succ_i$ variable is selected according to the pheromones $\tau_{i,j}$.



(c) Once the ending depot is reached, the ant starts with the route of the next vehicle. (d) All remaining nodes are assigned to the dummy vehicle (i.e., they are left unserved).

Fig. 2. Illustration of the graph traversal performed by one ant.

***τ* update.** After all n ants have produced a feasible solution (we call this set of solutions S_{upd}), their cost function value is used to make an update to the pheromone table. The update rule is the one described in [15] (adapted to be consistent with our conventions)

$$\tau_{i,j} = (1-\rho) \cdot \tau_{i,j} + \rho \cdot \sum_{s \in S_{upd}} \frac{F(s)}{\sum_{s' \in S_{upd}} F(s')} \tag{35}$$

where ρ is a learning rate that controls how fast the pheromones adapt to make the new solutions more likely and F is a *quality* function that in our case is defined as $F(s) = 1/cost(s)$. Note that *all* pheromones are also subject to a multiplicative evaporation of $1-\rho$.

5 Experimental Analysis

In this section we report and discuss the experimental analysis of the algorithms. The experimental setting is as follows.

For fair comparison, both the CP and the ACO+CP algorithms were implemented in Gecode (v 3.7.3) [16], the ACO variant consisting in specialized branching and search strategies.

All pheromones were initially set to $\tau_{\max} = 1$. The ρ parameter and the number of ants have been tuned by running an *F-Race* [17] with a confidence level of 0.95 over a pool of 210 benchmark instances from Citybike Vienna. Each instance, featuring a given number of stations, was considered with different number of vehicles ($V \in \{1,2,3,5\}$) and time budgets ($\hat{t} \in \{120, 240, 480\}$).

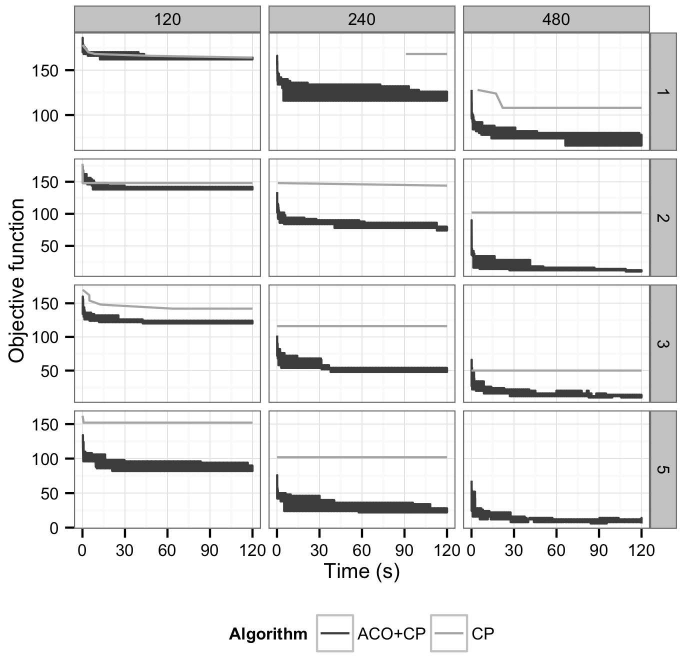


Fig. 3. Comparison between ACO+CP (dark, solid lines) and CP (light, thin lines) on a problem instance with 30 stations. The columns of the graph matrix represent the vehicle time budget and the rows represent the number of available vehicles.

Moreover, the algorithms were allowed to run for three different timeouts (30, 60, 120 seconds), totaling 7560 problems.

We tuned the number of ants $n \in \{5, 10, 15, 20\}$ and the learning rate ρ together, as we expected an interaction between the two parameters. The 8 candidate values for ρ were instead sampled from the low-discrepancy Hammersley point set in $[0.4, 0.8]$. This interval was chosen according to a preliminary tuning of the parameters, with $\rho \in [0, 1]$ and 32 samples. The result of the tuning process is that, for the considered set of problems, the best setup involves 5 ants and $\rho = 0.65$. All the experiments were executed on an Ubuntu Linux 12.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20GHz) cores.

Comparison between CP and ACO+CP. The main goal of this comparison is to understand if a dynamic branching strategy based on ACO can indeed outperform a static branching strategy. Figure 3 shows the results on an instance from the Citybike Vienna benchmark set featuring 30 stations. The choice of this instance has been driven by the fact that a time budget of 2 minutes was too low for CP to obtain even a single solution on larger instances.

The results of the comparison show that ACO+CP clearly outperforms the pure CP approach. In fact, the CP solver is declared significantly inferior by the F-Race procedure after just 15 iterations. The superior behavior of ACO+CP is confirmed also from the analysis reported in Figure 3, for the variants of a single problem instance with 30 stations. Note that the ACO+CP data is based on 5 repetitions of the same experiment, as the process is intrinsically stochastic.

It is possible to see that the cost values achieved by ACO+CP are always lower than those of CP and in one case (namely time budget 480 and 5 vehicles) CP is even not able to find a solution within the granted timeout despite the fact that it is somehow a loosely constrained instance.

Comparison with other methods. In this second experiment, we compare ACO+CP and CP, with state-of-the-art results of [1], who solved the same set of instances by a Mixed Integer Linear Programming solver (MILP) and a Variable Neighborhood Search (VNS) strategy. The results of the comparison are reported in Table 2, where we compare against the best of the three different VNS strategies described in [1]. The results reported are averages across instances with the same number of stations.

In this respect, the results are still unsatisfactory, since the best VNS approach of [1] is outperforming our ACO+CP on almost all instances. Nevertheless, our ACO+CP is able to do better than the MIP approach for mid- and large-sized instances.

6 Conclusions and Future Work

In this paper we tackle the problem of balancing the Citybike Vienna bike sharing system by means of a hybrid ACO+CP approach. The contributions of the paper are twofold.

First, we devise a novel CP formulation for the problem based on an extension of the classical CP vehicle routing model [12]. Up to the best of our knowledge, this is, together with [11], one of the two available CP formulations. Second, we propose a novel hybrid ACO+CP approach with the aim of improving the results of the pure CP solver. The proposed hybridization approach is quite general and can be applied also to other problems having a similar bi-level optimization structure. Moreover, the hybrid approach is implemented as an extension of the Gecode CP system and requires a small customization for handling different problem models.

From our experiments, it is clear that the ACO+CP approach outperforms the standard branch-and-bound CP solution method. However, despite these promising initial results, the performances of ACO+CP are still not as good as those achieved by the state-of-the-art metaheuristic approaches for this problem.

Among the alternatives we want to explore, there is the validation of the proposed ACO+CP approach on other bi-level optimization problems such as the integrated vehicle routing and packing problem. Moreover, we plan to investigate other methods for combining metaheuristics and CP, e.g., LNS.

Table 2. Comparison of our CP and ACO+CP solvers with the MIP and the best VNS approach of [1]

Instance			**CP**			**ACO+CP**			**MIP** [1]			**VNS** [1]	
S	V	$\hat{t}$	$\overline{obj}_{30}$	$\overline{obj}_{60}$	$\overline{obj}_{120}$	$\overline{obj}_{30}$	$\overline{obj}_{60}$	$\overline{obj}_{120}$	$\overline{ub}$	$\overline{lb}$	$\overline{time}$	$\overline{obj}$	$\overline{time}$
10	1	120	**28.3477**	**28.3477**	**28.3477**	28.5344	28.7477	28.5478	**28.3477**	28.3477	4	**28.3477**	2
10	1	240	14.0908	11.4915	9.5589	5.2276	4.7609	4.4810	4.2942	0.0424	3600	**4.2941**	10
10	1	480	14.8247	13.2922	9.8942	0.4322	0.9120	0.6052	0.0320	0.0276	3600	**0.0317**	17
10	2	120	10.2266	10.2266	10.2266	10.4001	10.6667	10.4268	**9.8269**	9.4768	911	9.9601	3
10	2	240	5.3652	4.6987	2.7662	0.4342	0.9274	0.1009	0.0340	0.0322	856	**0.0339**	19
10	2	480	5.3637	4.8971	3.2976	0.4854	0.4586	0.8584	**0.0317**	0.0313	1245	**0.0317**	15
20	2	120	72.4942	70.4279	68.7614	64.2558	62.9492	61.4561	**5.8294**	26.9012	3600	55.3628	8
20	2	240	74.2422	72.3754	71.5087	18.3904	17.3372	15.3907	19.7884	0.0383	3600	**4.2575**	58
20	2	480	74.1093	72.7093	72.5756	3.9748	2.7743	2.8943	1.8906	0.0403	3600	**0.0615**	142
20	3	120	67.5712	64.1051	61.5053	48.4287	47.1622	45.0557	37.3759	1.4770	3600	**31.7763**	13
20	3	240	74.3813	74.1811	74.1143	7.5511	5.8310	4.7641	6.2083	0.0401	3600	**0.0650**	65
20	3	480	74.3814	74.1811	74.1144	4.5878	2.5211	2.5874	13.4191	0.0316	3600	**0.0614**	114
30	2	120	127.5604	126.4939	125.5608	122.4552	119.2022	118.0823	106.9631	56.3908	3600	**104.7633**	12
30	2	240	117.2520	116.5857	116.3188	75.7637	73.2173	70.4309	74.9886	0.0487	3600	**34.6608**	109
30	2	480	101.8650	101.8650	101.4652	13.5173	11.1311	9.3847	69.8069	0.0432	3600	**0.0925**	491
30	3	120	–	117.7058	115.6393	107.7879	105.1748	102.0554	90.4419	16.6454	3600	**78.1773**	21
30	3	240	104.6052	104.4719	104.0054	46.0564	42.7502	40.5769	61.6715	0.0461	3600	**7.1523**	191
30	3	480	100.7422	100.6089	100.6089	–	10.7450	8.5595	175.4000	0.0015	3600	**0.0925**	399
60	3	120	–	–	–	307.8148	304.4283	300.2154	274.3101	157.7350	3600	**253.8462**	45
60	3	240	–	–	–	245.3374	238.1644	233.6585	370.2000	0.0000	3600	**126.8282**	521
60	3	480	205.8871	205.8871	205.8870	127.2744	122.7286	117.6223	–	–	3600	**6.7758**	3600
60	5	120	–	–	–	283.0537	278.0540	272.8145	289.3111	34.9784	3600	**196.6749**	99
60	5	240	–	–	–	184.7371	179.0572	173.6710	370.2000	0.0000	3600	**41.6161**	1556
60	5	480	–	–	–	–	–	–	–	–	3600	**0.1902**	3600
90	3	120	–	–	–	511.8807	507.2943	504.2013	492.2319	290.8990	3600	**441.6473**	82
90	3	240	–	–	–	451.7232	445.4705	438.2044	566.2667	0.0000	3600	**294.5646**	985
90	3	480	–	–	–	334.6610	326.4350	319.5826	–	–	3600	**101.1221**	3600
90	5	120	–	–	–	490.3193	480.7739	473.9345	566.2667	0.0000	3600	**376.1432**	169
90	5	240	–	–	–	393.4433	383.3375	375.5915	–	–	3600	**174.3566**	3304
90	5	480	–	–	–	213.3140	202.3017	192.3832	–	–	3600	**1.6855**	3600

Acknowledgements. This work is part of the project BBSS, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic programme I2VSplus under grant 831740 and it has been also supported by Google Inc. under the Google Focused Grant Program on "Mathematical Optimization and Combinatorial Optimization in Europe". We thank Marian Rainer-Harbach, Petrina Papazek, Bin Hu and Günther R. Raidl from the Vienna University of Technology, and Matthias Prandtstetter and Markus Straub from the Austrian Institute of Technology, and City Bike Vienna for the collaboration in this project, constructive comments and for providing the test instances and the results of the methods they developed.

References

1. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In Middendorf, M., Blum, C.,

eds.: Evolutionary Computation in Combinatorial Optimization. Volume 7832 of LNCS, Springer Berlin–Heidelberg (2013) 121–132

2. Meyer, B., Ernst, A.: Integrating aco and constraint propagation. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L., Mondada, F., Stützle, T., eds.: Ant Colony Optimization and Swarm Intelligence. Volume 3172 of LNCS. Springer Berlin–Heidelberg (2004) 166–177
3. Khichane, M., Albert, P., Solnon, C.: CP with ACO. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer (2008) 328–332
4. Khichane, M., Albert, P., Solnon, C.: Strong combination of ant colony optimization with constraint programming optimization. In Lodi, A., Milano, M., Toth, P., eds.: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems—CPAIOR 2010. Volume 6140 of LNCS, Berlin–Heidelberg, Germany, Springer (2010) 232–245
5. Khichane, M., Albert, P., Solnon, C.: Integration of ACO in a constraint programming language. Ant Colony Optimization and Swarm Intelligence (2008) 84–95
6. Pisinger, D., Ropke, S.: Large neighborhood search. In Gendreau, M., Potvin, J.Y., eds.: Handbook of Metaheuristics. Volume 146 of International Series in Operations Research & Management Science. Springer US (2010) 399–419
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) (2011) 37–61
8. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Technical Report CIRRELT-2012-09, CIRRELT, Montreal, Canada (2012) submitted to Transportation Science.
9. Raviv, T., Tzur, M., Forma, I.A.: Static Repositioning in a Bike-Sharing System: Models and Solution Approaches. EURO Journal on Transportation and Logistics, doi:10.1007/s13676-012-0017-6 (2012)
10. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. To appear in Discrete Optimization (2012)
11. 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)
12. Kilby, P., Shaw, P.: Vehicle routing. In Rossi, F., Beek, P.v., Walsh, T., eds.: Handbook of Constraint Programming. Elsevier, New York, USA (2006) 799–834
13. Dorigo, M., Birattari, M.: Ant colony optimization. In: Encyclopedia of Machine Learning. (2010) 36–39
14. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, USA (2006)
15. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **34**(2) (April) 1161–1172
16. Gecode: Generic constraint development environment (2006) Available from `http://www.gecode.org`.
17. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. Experimental methods for the analysis of optimization algorithms (2010) 311–336