Bike sharing system: solving the static rebalancing problem

Daniel Chemla, Frédéric Meunier, Roberto Wolfler Calvo

BIKE SHARING SYSTEM: SOLVING THE STATIC REBALANCING PROBLEM

DANIEL CHEMLA, FRÉDÉRIC MEUNIER, AND ROBERTO WOLFLER CALVO

Abstract. This paper deals with a new problem that is a generalization of the many to many pickup and delivery problem and which is motivated by operating self-service bike sharing systems. There is only one commodity, initially distributed among the vertices of a graph, and a capacitated single vehicle aims to redistribute the commodity in order to reach a target distribution. Each vertex can be visited several times and also can be used as a buffer in which the commodity is stored for a later visit. This problem is NP-hard, since it contains several NP-hard problems as special case (the TSP being maybe the most obvious one). Even finding a tractable exact formulation remains problematic.

This paper presents efficient algorithms for solving instances of reasonable size, and contains several theoretical results related to these algorithms. A branch-and-cut algorithm is proposed for solving a relaxation of the problem. An upper bound of the optimal solution of the problem is obtained by a tabu search, which is based on some theoretical properties of the solution, once fixed the sequence of the visited vertices. The possibility of using the information provided by the relaxation receives a special attention, both from a theoretical and practical point of view. It is proved that to build a feasible solution of the problem by using the one obtained by the relaxation is an NP-hard problem. Nevertheless, a tabu search initialized with the optimal solution of the relaxation often shows that it is the optimal one.

The algorithms have been tested on a set of instances coming from the literature, proving their effectiveness.

1. Introduction

Nowadays self-service bike sharing systems are flourishing all over the world. The exploitation of such transport systems implies various problems and one of them is to ensure people that they will be able to find a bike or to park it at each station all the day long. Therefore, a regulation system is necessary for maintaining a prefixed optimal number of bikes at each station in order to fulfill at best the demand. It is done with a fleet of vehicles which are driven around the city and move bikes from a place to another. This work restrains to the rebalancing problem with one vehicle and in the *static* case. Static means that users cannot act on the bikes during the rebalancing process. The static rebalancing problem can be classified as a Single Vehicle One-commodity Capacitated Pickup and Delivery Problem (SVOCPDP). This problem is the one faced by an operator during the night when the number of moving bikes is negligible and when the city is divided into districts. Each district is covered by a single truck that has to redistribute the bikes in order to respond to the morning peak at best. This is a realistic problem since it is during the night that the impact of the regulation is the most important, as explained by operators. There are even such systems (for instance the one in Lyon, France), which are closed during the night, to make this regulation task easier.

To get an idea of the size of such a system, the numerical features of the *Vélib* system in Paris are presented. This system offers more than $20'000$ bikes deployed in about $1'200$ stations that are in Paris and its border cities and a fleet of twenty three trucks of capacity equal to 20 are used to move bikes during the day to match the demand. If the city is divided into areas on which only one truck operates, then each truck has to work on about 50 stations. This paper focus on this part of this operating problem: how to deal with a part of the city assigned to a single vehicle. The multiple vehicles case is currently under study (see [CMWC11]).

The problem can be formalized as follows. Let $G = (V, A)$ be a complete directed graph where $V = \{0, \ldots, n\}$ is the vertex set composed by $n + 1$ vertices, the vertices in $\{1, \ldots, n\}$ representing the stations and the vertex 0 representing the depot, and where A is the set of arcs. For each arc $(i, j) \in A$, we denote by $c_{(i,j)}$ the cost of the arc (i, j). The cost is assumed to satisfy the triangular inequality (i.e., $c_{(i,j)}+c_{(j,k)} \geq c_{(i,k)}$ for all $i, j, k \in V$). Each vertex i has a capacity $C_i \in \mathbb{Z}_+$. For each vertex $i \in V$, its initial state in bikes is defined by $p_i \in \mathbb{Z}_+$ and its target, or final, state by $q_i \in \mathbb{Z}_+$. A vertex is *in excess* (resp. *in default*) if $p_i > q_i$ (resp. $p_i < q_i$). Some vertex can be *initially balanced* i.e. $p_i = q_i$. Moreover, throughout the paper the

imbalance $d_i = p_i - q_i$ is used and the depot is always assumed to have no bike: $C_0 = p_0 = q_0 = 0$ and $d_0 = 0$. The vehicle has also a capacity Q.

A feasible solution for the SVOCPDP, also called a *route*, is a sequence of vertices, starting and finishing with the depot 0, together with bike displacements within the limits of capacity constraints, at the end of which the system is balanced: each vertex i has been brought from its initial state p_i to its target state q_i. In this case, the sequence of vertices is said to be *induced* by the route. The goal of the SVOCPDP is to find the minimal cost route. Note that the convergence for each vertex from p_i to q_i is not required to be monotonous: drops and multiple visits of the vertices are allowed, i.e. bikes can be loaded from vertices in default or unloaded at vertices in excess and transfers can take place at initially balanced vertices. Figure 1 shows an example of instance with 9 vertices (the depot and 8 stations) including one initially balanced vertex. A pair of values representing (p_i, q_i) is displayed next each vertex and the capacity of the vehicle is equal to 8. Figure 2 shows a feasible solution.

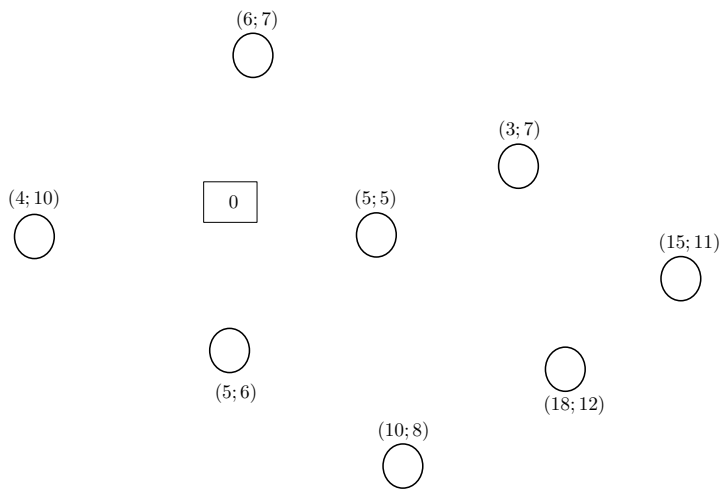

FIGURE 1. Example of an instance.

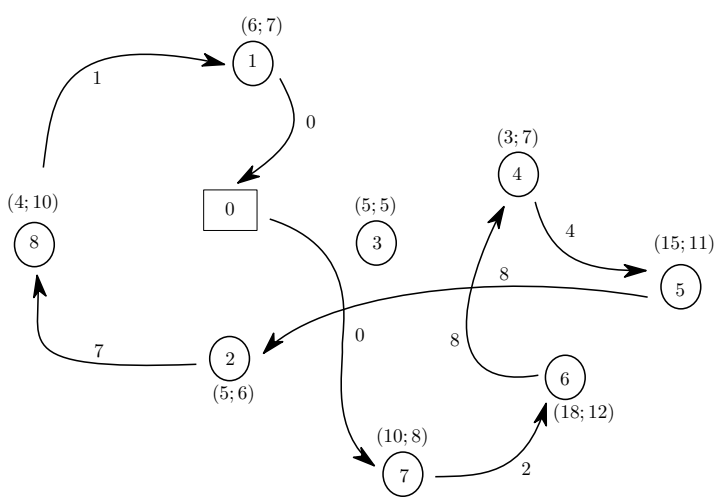

FIGURE 2. Example of a feasible solution (i.e. a route)

1.1. **Complexity.** The SVOCPDP is NP-hard since it contains NP-hard problems as special cases. It is easy to see it, but details are given for sake of completeness. The TSP is obviously one of them. Set $p_i = 0$ and $q_i = 1$ for all vertices $i \in \{1, \ldots, n-1\}$, and set $p_n = n-1$ and $q_n = 0$. Add a depot at distance 0 from the vertex n. Set $Q = n-2$. The optimal solution of the SVOCPDP coincides with the optimal TSP solution.

The 2-partition is another special case. Let $b_1, \ldots, b_n$ be n non-negative integers such that $\sum_i b_i$ is even. Define $m = \frac{1}{2}\sum_{i=1}^{n} b_i$. The 2-partition problem is a decision problem asking whether there exists $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} b_i = m$. Take the complete graph with $n+3$ vertices: n vertices with $p_i = b_i$ and $q_i = 0$; 2 vertices with $p_i = 0$ and $q_i = m$ and the depot. Define the $c_{(i,j)}$ to be 1 for each arc (i,j) and the capacity of the vehicle equal to m. The optimal solution of the SVOCPDP problem is equal to $n+3$ if and only if there is a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} b_i = m$.

Moreover, an optimal route of the SVOCPDP problem may not have an encoding that is polynomial in the size of the input. The simple case with three vertices – the depot 0, one vertex 1 with $p_1 = B$ and $q_1 = 0$ and one vertex 2 with $p_2 = 0$ and $q_2 = B$ – is enlightening. Assuming that $Q = 1$, the optimal sequence of vertices is $0, 1, 2, 1, 2, \ldots, 0$, with a number of terms $= 2B + 2$, although the input is in $O(\log_2 B)$.

1.2. **Main contributions of this paper.** This paper deals with a new problem (see [BBC+11] for a first reference on this problem with theoretical results such as approximation algorithms or special polynomial cases). We emphasize that we are not only interested by efficient algorithms for solving it, but also by theoretical results linked with these algorithms and giving a better understanding of the problem.

An exact model is presented, which does not seem to be tractable. Therefore, a relaxation, which turns out to be an integer program with an exponential number of constraints, is proposed and solved by a branch-and-cut algorithm. The optimal solution of this relaxation generally provides a good lower bound of the optimal solution of the original problem (the quality of this lower bound is proven experimentally). For a fixed sequence of vertices visited by the vehicle, we prove that it is possible to find in polynomial time the operations that bring the system to the possible state nearest to the target state (Propositions 1 and 2). In particular, we prove that it is possible to decide in polynomial time whether there is a feasible solution inducing precisely a fixed sequence of visited vertices. It allows to derive a purely combinatorial tabu search providing upper bounds for the problem: the underlying neighborhood does not take care of the loading of the vehicle. The tabu search proves good behaviors.

Moreover, a special attention is given to the way of using the information provided by the relaxation: both from a theoretical point of view and from a practical one. From a theoretical point of view, we are able to prove that even if the relaxation is quite natural and provides (experimentally) good lower bounds, it is an NP-complete problem to decide whether there is an optimal solution of same cost. More precisely, given the number of times each arc is traversed, and given the number of bikes carried on each visit on each arc, it is an NP-complete problem to decide whether there is an optimal solution realizing these numbers. This property is formalized in Propositions 5–8. From a practical point of view, it is experimentally shown that the solutions obtained with the relaxation are often the optimal ones. Indeed, the tabu search is in general able to find a solution of almost the same cost, when initialized with the optimal solution of the relaxation.

1.3. **Notations and basic notions.** We define for all subsets $S \subseteq V$:

- $\bar{S} = V \backslash S$
- $\delta^+(S) := \{(i,j) \in A : i \in S; j \in \bar{S}\}$
- $\delta^-(S) := \{(i,j) \in A : i \in \bar{S}; j \in S\}$
- $\delta(0) := \delta^+(\{0\}) \cup \delta^-(\{0\})$
- $d(S) = \sum_{j \in S} d_j$
- $\mu(S)$ is equal to 1 whenever there is at least one initially non-balanced vertex in S, 0 otherwise.

Given $z \in \mathbb{R}_+^A$, $G[z]$ is the directed graph obtained from G by deleting the arcs a with $z_a = 0$. This graph is called the *support graph* of z.

A notion used several times in the paper is the one of a *b-flow*. A b-flow is an usual notion in combinatorial optimization (see for instance [Sch03, KV02]). Given a directed graph $D = (U, A')$, a value $b \in \mathbb{R}^U$, and capacities $l, u \in \mathbb{R}^{A'}$ with $l \leq u$, a b-flow is a map $f : A' \to \mathbb{R}$ such that $l_a \leq f(a) \leq u_a$ for all $a \in A'$ and $\sum_{a \in \delta^+(v)} f(a) = b_v + \sum_{a \in \delta^-(v)} f(a)$ for all $v \in U$. If it exists, a b-flow can be computed in strongly

polynomial time. Moreover, when all the b_v and the l_a, u_a are integral, if a b-flow exists, there is an integral one.

1.4. **Plan.** In Section 2, differences between the SVOCPDP and problems in the literature are presented. Section 3 presents the aforementioned proposition that enables to find in polynomial time the operations that bring the system to the possible state nearest to the target state, given a fixed sequence of vertices visited by the vehicle. An exact model of the problem is given in Section 4, and a relaxation is presented in Section 5. In Section 6, we prove that deciding whether a solution of the relaxation is a feasible solution for the SVOCPDP is NP-complete. The Section 7 contains the description of the branch-and-cut algorithm that is used to solve the relaxation. The proposition of Section 3 is used in Section 8 for deriving a tabu search. Finally, Section 9 presents the computational results on instances from the literature with a slight adaptation in order to fit the SVOCPDP's features.

2. Literature review

In the literature, similar problems are the One-Commodity Pickup-and-Delivery TSP (1PDTSP) studied by [HPSG04a] and the Swapping Problem defined by [AH92]. This latter has been solved by [BGG08, BGL09] in its preemptive and its non-preemptive versions. For solving the 1PDTSP, several papers have been published recently in the literature. They propose different exact and heuristic algorithms (see [HPSG04b], [HPSG07], [HPRMSG09] and [HIMU12]). Since these two problems present several similarities with our problem, more details about them are given in the next paragraph. A recent conference paper by [RTF09] is motivated by the same application. They discuss different variants of the problem of repositioning the bikes which are modeled by mixed linear programs and solved using CPLEX. For some variants, they are able to solve several instances to optimality (up to 60 stations and 2 vehicles for their so-called "Arc-Indexed" variant). Our model is close to their "Sequence-Indexed" variant (see Section 3.4 of [RTF09]), but instead of computing a minimum cost route with fixed target states, they try to find the best repartition of bikes that can be achieved by a fleet of one or several vehicles within a time limit. Moreover, in the Sequence Index formulation given by [RTF09], drops are not allowed, a thing that will be fixed in an upcoming version by [RT11].

The SVOCPDP gathers aspects from both the Swapping Problem and the 1PDTSP but differs by main features. In the Swapping Problem a single vehicle has to move unitary objects from a vertex to another. There are m different types of objects and the vehicle has a unitary capacity: it can only move one object at the time. [AH92] showed that a vertex could be visited at most three times in the optimal solution. This theorem was the starting point of the work of [BGG08, BGL09] for solving the Swapping Problem with a branch-and-cut algorithm. The SVOCPDP is different since there is only one type of object (bikes), but the supply and the demand are greater than one and the vehicle capacity is Q. Anily and Hassin's theorem does not hold anymore: for example in the trivial instance with one pickup vertex with $p_i = 5Q$ and $q_i = 0$ and one delivery vertex with $p_i = 0$ and $q_i = 5Q$, the vehicle has to do five round trips between the two vertices.

The main difference with the 1PDTSP is that in the SVOCPDP a vertex can be visited several times, whereas in the 1PDTSP the solution is required to be an Hamiltonian cycle. In the 1PDTSP, the depot is assumed to have an unlimited number of objects, but this fact can be modelled in the SVOCPDP by a vertex at distance 0 from the depot with Q bikes available and a target state of Q bikes. A solution of the 1PDTSP is a feasible solution for the SVOCPDP, but the SVOCPDP can have a lower-cost optimal solution. Moreover, the 1PDTSP may have instances without feasible solutions (the instance of Figure 3, with $Q = 3$, is an example), something that can not happen for the SVOCPDP.

The fact that one can get better solutions in routing problems when customers are allowed to be visited several times has already be noticed in other works. Note for instance the work by [ASH06], in which a Split Delivery Problem is solved through a tabu search. In the SVOCPDP, any vertex can be used as a buffer where bikes can be indifferently temporary loaded or unloaded before being moved to their final destination. In particular, initially balanced vertices are not required to be visited, but may act as temporary depot in some optimal solutions.

The buffers can improve the optimal solution in some instances such as shown in the example given in Figure 3 and in Figure 5. In Figure 3 the square with the 0 inside is the depot and its distance with vertex 1 is null. For any other vertex i, $(p_i; q_i)$ is given. The capacity of the vehicle is $Q = 3$ and the distances

between all the peripheral vertices to the central one is 1. We show here the optimal solution, whose value is equal to 10. In this solution a bike is temporary unloaded at the central vertex 6 and then reloaded on the vehicle. The optimal solution would be equal to 12, if drops would be forbidden, because one of the odd number vertices would have to be visited twice.

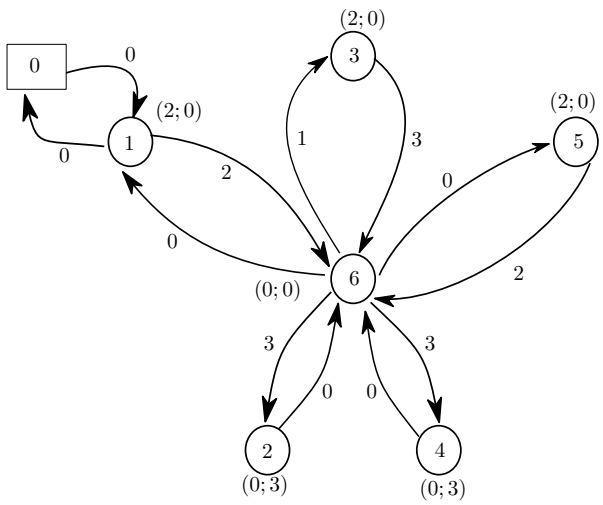

FIGURE 3. Drops can help optimality

Figures 4 and 5 present a situation where the non-monotonous convergence of the load of the vertices improves the solution. Figure 4 shows the best solution of the 1PDTSP. The two first vertices with initial and target states $(Q; 0)$ and $(0; Q)$ (Q being the capacity of the vehicle) are here to prevent a use of the unlimited number of bikes in an optimal solution, available in the depot for the 1PDTSP. Figure 5 represents the optimal solution for the SVOCPDP. In both figures, the square with 0 is the depot, as in Figure 3. In Figure 5, the vehicle takes a bike from the left-corner vertex of the square, increasing momentary the deficit in bikes, before coming back with the two missing bikes. With Euclidean distances, the solution of the SVOCPDP is better than the one of the 1PDTSP.

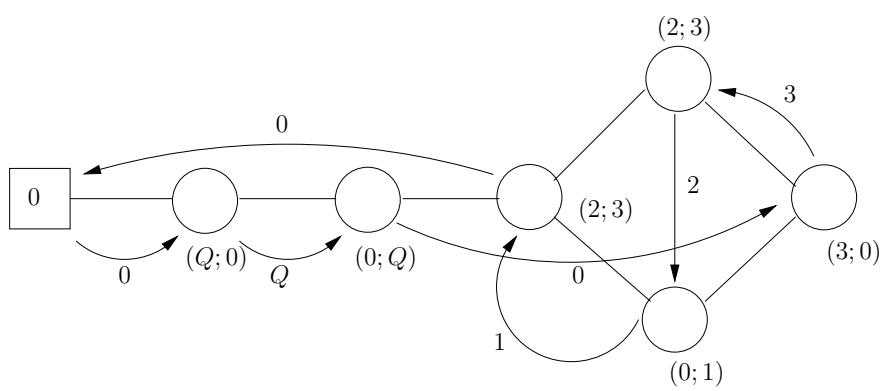

FIGURE 4. An optimal solution for the 1PDTSP

3. Dealing with sequences and routes

The difficulty of the SVOCPDP is that a feasible solution is identified by both a sequence of vertices and a set of numbers of bikes carried on arcs. The following proposition (and its companions Propositions 2

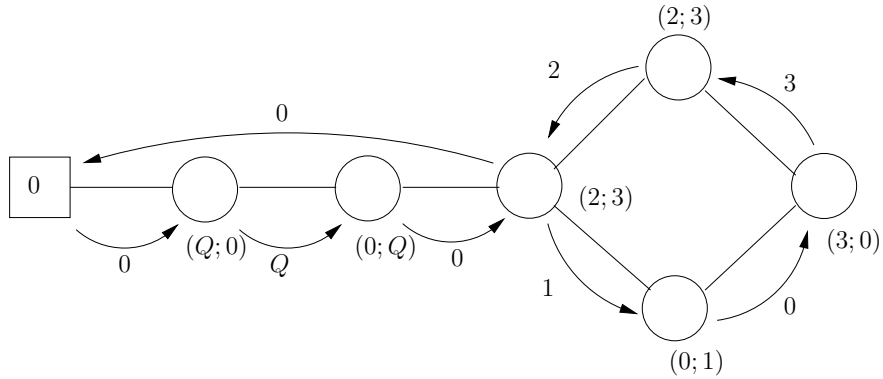

FIGURE 5. An optimal solution for the SVOCPDP with the same input as for Figure 4, the non-monotonous convergence helps

and 3) enables us to work only with sequences of vertices. It will be particularly useful in Section 8 when we will design a local search for the SVOCPDP.

Proposition 1. *Let $i_1, i_2 \ldots, i_k$ be a sequence of vertices, starting and finishing at the depot $0 = i_1 = i_k$. There is a polynomial algorithm finding new initial and target states (p'_i, q'_i) for each vertex i and a route inducing this sequence of vertices and satisfying these new states such that*

- *$0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i*
- *$\sum_i p'_i = \sum_i q'_i$*
- *the quantity $\sum_{i \in V} p'_i$ is maximal.*

In particular, it is possible to decide in polynomial time whether a sequence of vertices is induced by a feasible solution (in this later case, $p'_i = p_i$ for each vertex i).

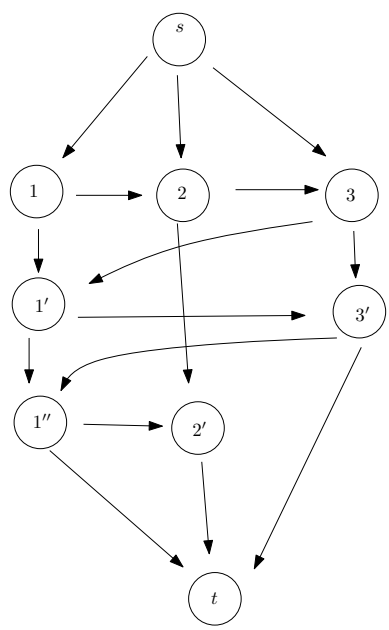

FIGURE 6. Example of the graph used in the algorithm of Proposition 1

The proposition says roughly speaking that it is possible to find the best bike displacements compatible with the sequence of vertices. The quantity $\sum_{i \in V}(q_i - q'_i)$ can be interpreted as a kind of "degree of infeasibility" of a sequence and the proposition shows how to minimize it polynomially.

Proof. Let us build a directed graph $D = (U, A')$ as follows. U has $k+2$ elements: each vertex i_j (make as many copies of a vertex i of G as there are occurrences of i in the sequence) and two more vertices s and t. The arcs in A' are of four types:

(1) one arc between s and the first occurrence of each vertex i in the sequence, with capacity p_i,
(2) one arc (i_j, i_{j+1}) for each $j = 1, \ldots, k-1$, with capacity Q,
(3) one arc between the rth occurrence of each vertex i and its $(r+1)$th occurrence, if there is one, with capacity C_i,
(4) one arc between the last occurrence of each vertex i in the sequence and t, with capacity q_i.

See Figure 6 for an illustration with the sequence $0 \to 1 \to 2 \to 3 \to 1 \to 3 \to 1 \to 2 \to 0$. Computing a maximum s-t flow in this graph leads to the proposition. Indeed, any s-t flow on D encodes possible bike displacements compatible with the given sequence of vertices. The numbers of bikes to be moved while going from i_j to i_{j+1} are given by the flow on arc (i_j, i_{j+1}) (arcs defined in 2.); the number of bikes remaining in a vertex i after the rth visit of the vehicle is given by the flow on arc between the rth occurrence of vertex i and its $(r+1)$th occurrence (arcs defined in 3.); the initial and final numbers of bikes in a vertex i are given respectively by the flows on arcs defined in 1. and 4. And conversely, any bike displacements compatible with the given sequence of vertices induce an s-t flow.

p'_i is then the value of the flow in the arc between s and the first occurence of i, and q'_i the value of the flow in the arc between the last occurence of i and t. If a vertex i is not present in the sequence, then we set $p'_i = q'_i = \min(p_i, q_i)$. □ □

In the case that all C_i are sufficiently large, Proposition 1 has a nice and maybe quite unexpected corollary, formalized by the following proposition. If we are not allowed to change the initial states, but only the final ones, the solution given by Proposition 1provides a route with new final states closest to the original ones.

Proposition 2. *Assume that we have $C_i = +\infty$ for all vertices i. Let $i_1, i_2 \ldots, i_k$ be a sequence of vertices, starting and finishing at the depot $0 = i_1 = i_k$. Consider the problem of finding bike displacements along this sequence leading to the final state $\tilde{q}$ closest to q when starting from the initial state p, where "closest" means the state that minimizes the L_1 norm $||\tilde{q} - q||_1 = \sum_{i \in V} |\tilde{q}_i - q_i|$. The bike displacements given by Proposition 1 for this sequence are precisely the solution of this problem.*

Proof. See Appendix. □ □

As a by-product of the proof of Proposition 1, we get

Proposition 3. *Assume that we have a feasible solution to the SVOCPDP with fractional numbers of bikes carried during some moves. Then there is also a feasible solution of the same cost with only integral numbers of bikes carried during the moves.*

Thanks to this proposition, we can "forget" the integrality constraint without changing the cost of the optimal solution.

4. An exact model

The exact model is based on the assumption that a constant β_i is given for each vertex i, which is an upper bound of the number of times the vehicle has to visit i in any optimal solution. For instance, given a feasible solution of total cost O, we can set $\beta_i := O / \min_{j \neq i} c_{(i,j)}$ ([Min10]). The model is intractable with a direct approach, since the β_i computed in this way can be quite huge and since the model involves variables with four indices. It would be interesting to compute a tighter β_i, which would reduce the size of the exact model, but whether it is possible from *a priori* considerations remains an open question.

We introduce the following variables. $x_{i,t}$ takes the value 1 only if vertex i is visited at least t times. $z_{i,t,i',t'}$ takes the value 1 only if the vehicle visits vertex i' for the t'th time just after having visited i for the tth time, and $y_{i,t,i',t'}$ is then the number of bikes that is carried by the vehicle during this move. The variables y are not required to be integral since, according to Proposition 3, it does not change the optimal value of the linear program. $u_{i,t,i',t'}$ is a variable which takes the value 1 if the t'th visit of vertex i' comes in the route after the tth time of vertex i (but not necessarily right after).

$$(P) \quad z(P) = \min \sum_{(i,i')\in A} \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} c_{(i,i')} z_{i,t,i',t'}$$

$$s.t.$$

$$x_{i,t} \leq x_{i,t-1} \qquad \forall i \in V,\ \forall t \in \{2,\dots,\beta_i\} \tag{1}$$

$$\sum_{i'\in V\setminus\{i\}} \sum_{t'=1}^{\beta_{i'}} z_{i,t,i',t'} = x_{i,t} \qquad \forall i \in V,\ \forall t \in \{1,\dots,\beta_i\} \text{ except for } (i,t)=(0,2) \tag{2}$$

$$\sum_{i'\in V\setminus\{i\}} \sum_{t'=1}^{\beta_{i'}} z_{i',t',i,t} = x_{i,t} \qquad \forall i \in V,\ \forall t \in \{1,\dots,\beta_i\} \text{ except for } (i,t)=(0,1) \tag{3}$$

$$u_{i,t,i',t'} \geq u_{i,t,i'',t''} + z_{i'',t'',i',t'} - 1 \qquad \forall i,i',i'' \in V, \forall t \in \{1,\dots,\beta_i\},\ \forall t' \in \{1,\dots,\beta_{i'}\},\ \forall t'' \in \{1,\dots,\beta_{i''}\} \tag{4}$$

$$u_{i,t,i',t'} \geq z_{i,t,i',t'} \qquad \forall i,i' \in V, \forall t \in \{1,\dots,\beta_i\},\ \forall t' \in \{1,\dots,\beta_{i'}\} \tag{5}$$

$$u_{i,t,i,\tau} = 0 \qquad \forall i \in V, \forall t,\tau \in \{1,\dots,\beta_i\} \text{ with } \tau \leq t \tag{6}$$

$$y_{i,t,i',t'} \leq Q z_{i,t,i',t'} \qquad \forall i,i' \in V, \forall t \in \{1,\dots,\beta_i\},\ \forall t' \in \{1,\dots,\beta_{i'}\} \tag{7}$$

$$0 \leq \sum_{i'\in V\setminus\{i\}} \sum_{\tau=1}^{t} \sum_{\tau'=1}^{\beta_{i'}} (y_{i',\tau',i,\tau} - y_{i,\tau,i',\tau'}) + p_i \leq C_i \qquad \forall i \in V,\ \forall t \in \{1,\dots,\beta_i\} \tag{8}$$

$$\sum_{i'\in V\setminus\{i\}} \sum_{\tau=1}^{\beta_i} \sum_{\tau'=1}^{\beta_{i'}} (y_{i',\tau',i,\tau} - y_{i,\tau,i',\tau'}) + p_i = q_i \qquad \forall i \in V \tag{9}$$

$$z_{i,t,i',t'}, u_{i,t,i',t'} \in \{0,1\} \qquad \forall i,i' \in V, \forall t \in \{1,\dots,\beta_i\}, \forall t' \in \{1,\dots,\beta_{i'}\} \tag{10}$$

$$y_{i,t,i',t'} \in \mathbb{R}_+ \qquad \forall i,i' \in V, \forall t \in \{1,\dots,\beta_i\}, \forall t' \in \{1,\dots,\beta_{i'}\} \tag{11}$$

$$x_{i,t} \in \{0,1\} \qquad \forall i \in V, \forall t \in \{1,\dots,\beta_i\} \tag{12}$$

A way to see the exactness of this formulation consists in introducing a new graph, G^β, obtained by making β_i copies of each vertex i of G, and whose arcs link any pair of copies of distinct vertices. The idea is similar to the one for dynamic flows, for which one introduces usually *time-expanded graph*. A vertex of G^β is of the form (i,t) with $i \in V$ and $t \in \{1,\dots,\beta_i\}$ and any arc of the form $((i,t),(i',t'))$, with $i \neq i'$. A vertex (i,t) in G^β models the tth passage of the vehicle on vertex i. A feasible solution of the SVOCPDP is equivalent to an elementary directed path in G^β, starting from $(0,1)$, finishing at $(0,2)$ and visiting (i,t) only if all (i,τ) with $\tau < t$ have been visited.

(P) models this equivalent version on G^β. Constraints (1) are logical constraints implied by the definition of $x_{i,t}$. Constraints (2) and (3) ensure that we get in G^β an elementary path starting at $(0,1)$ and finishing at $(0,2)$. At first glance, there might be circuits as well. Constraints (4), (5), and (6) ensure that the solution is coherent with the t index: the tth visit of vertex i comes after the τth visit, for any $\tau < t$. As a direct by-product, we get also that there is actually no circuit.

Constraints (7) limit the number of bikes transshipped on an arc at each move to the capacity of the vehicle. Constraints (8) bound the number of bikes parked at a vertex i between 0 and its maximal capacity C_i at any time step. Constraints (10), (11) and (12) ensure that the vertices have all reached their target state at the end of the route.

Remark. The coherence on the t indices implied by the variables $u_{i,t,i',t'}$ and the constraints (4), (5) and (6) in (P) can alternatively be obtained by the introduction of variables $h_{i,t} \in \{1,\dots,\sum_{i=1}^n \beta_i\}$ encoding the instant of tth passage on vertex i and the use of "big-M" constraints, in a similar spirit as for the TSP with time windows (see for instance [DL91]).

5. Relaxations

This section presents two equivalent mixed integer programs. They represent a relaxation of the original problem, since even solved to optimality they produce a lower bound of the optimal solution.

5.1. **A first relaxation.** Using the variables of the exact model (P) of Section 4, a relaxation for the SVOCPDP can be obtained by defining $z_{(i,i')} = \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} z_{i,t,i',t'}$, which represents the number of times

the arc $(i,i') \in A$ is traversed in the solution, and by defining $y_{(i,i')} = \sum_{t=1}^{\beta_i} \sum_{t'=1}^{\beta_{i'}} y_{i,t,i',t'}$, which represents the total number of bikes transported on it.

The relaxation is

$$(RP1) \quad z(RP1) = \min \sum_{(i,j)\in A} c_{(i,j)} z_{(i,j)} \tag{13}$$

$$s.t.$$

$$\sum_{j\in V} z_{(i,j)} = \sum_{j\in V} z_{(j,i)} \qquad \forall i \in V \tag{14}$$

$$\sum_{i\in V\setminus\{0\}} z_{(0,i)} = 1 \tag{15}$$

$$\sum_{i\in V\setminus\{0\}} y_{(0,i)} = 0 \tag{16}$$

$$p_i + \sum_{j\in V\setminus\{i\}} y_{(j,i)} = q_i + \sum_{j\in V\setminus\{i\}} y_{(i,j)} \qquad \forall i \in V \tag{17}$$

$$\sum_{(i,j)\in\delta^+(S)} z_{(i,j)} \geq \mu(S) \qquad S \subseteq V \setminus \{0\} \tag{18}$$

$$0 \leq y_{(i,j)} \leq Q z_{(i,j)} \qquad \forall (i,j) \in A \tag{19}$$

$$z_{(i,j)} \in \mathbb{Z}_+, \qquad \forall (i,j) \in A \tag{20}$$

Again, requiring that the $y_{(i,j)}$ are integral does not improve the value of the relaxation (RP1). Indeed, whenever the values of the $z_{(i,j)}$ are fixed, the variables $y_{(i,j)}$ are solutions of a b-flow problem, and hence, since the p_i, q_i and Q are integer numbers, there is an optimal solution with integral values for the $y_{(i,j)}$.

Constraints (14) and (15) come from the constraints (2), and (3). Constraints (16), (17) and (19) come from the constraints (7) and (9). Finally, constraints (18) ensure the connectivity of the solution, while initially balanced vertices might be skipped in an optimal solution.

The proposed relaxation does not take into account the evolution on the number of bikes on each vertex at each time-step: all the moves are considered simultaneously and so the sequential dimension of the problem disappears. Therefore, although any solution of the SVOCPDP provides a solution of (RP1), a solution of model (RP1) might not be a solution of the SVOCPDP, as shown by the example given in Figure 7. In this example the values $(p_i; q_i)$ are displayed next to each vertex, $Q \geq 2$ and the optimal solution of (RP1) is represented: each arc a in the figure corresponds to $z_a = 1$, the others z_a being equal to 0 and the numbers near each arc are the y_a. The optimal solution satisfies model (RP1), but violates SVOCPDP: the vehicle would have to take one bike from the first vertex that is not yet arrived there.

However, the solution of (RP1) is often a solution to the SVOCPDP as illustrated in the computational results section (Section 9).

5.2. **A second relaxation.** The former mixed integer program requires two families of variables $z_{(i,j)}$ and $y_{(i,j)}$. Linear program solvers are sensitive to the number of variables, and even if the $y_{(i,j)}$ can be assumed to be real, (RP1) is too big to be solved by standard solvers. A way to get a tractable formulation consists in reducing the number of variables. Therefore, we now define an integer program which contains only the $z_{(i,j)}$ variables and we show that solving the new integer program is equivalent to solving the former one.

$$(RP2) \quad z(RP2) = \min \sum_{(i,j)\in A} c_{(i,j)} z_{(i,j)} \tag{21}$$

$$s.t.$$

$$\sum_{j\in V} z_{(i,j)} = \sum_{j\in V} z_{(j,i)} \qquad \forall i \in V \tag{22}$$

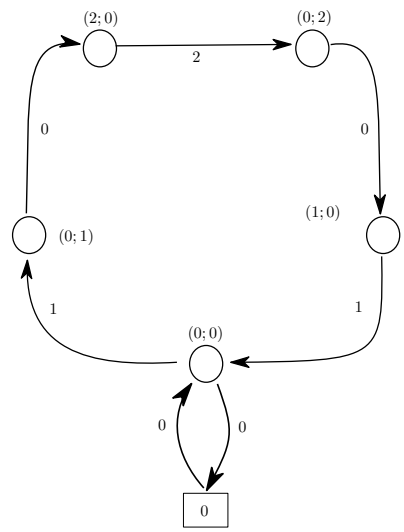

FIGURE 7. A solution of (RP1) that is not a solution of the SVOCPDP

$$\sum_{i \in V \setminus \{0\}} z_{(0,i)} = 1 \tag{23}$$

$$\sum_{(i,j) \in \delta^+(S)} z_{(i,j)} \geq \mu(S), \qquad \forall S \subseteq V \setminus \{0\} \tag{24}$$

$$\sum_{(i,j) \in \delta^+(S) \setminus \delta(0)} z_{(i,j)} \geq \left\lceil \frac{d(S)}{Q} \right\rceil \qquad \forall S \subseteq V \tag{25}$$

$$z_{(i,j)} \in \mathbb{Z}_+, \qquad \forall (i,j) \in A \tag{26}$$

Any feasible solution of the SVOCPDP induces $z_{(i,j)}$ that satisfy the constraints of program (RP2). Constraints (22), (23) and (24) are similar to those of the first relaxation (RP1). Constraints (25) are the capacity constraints requiring that, for any subset $S \subseteq V$, the vehicle goes at least $\left\lceil \frac{|d(S)|}{Q} \right\rceil$ times into S. The absolute value $|\cdot|$ is useless in (RP2). Indeed, if $d(S) \geq 0$, it is not necessary. If $d(S) < 0$, (25) are redundant and in anyway, when (25) is written with $\bar{S}$, we obtain precisely what would have been obtained with the $|.|$ for S.

Constraints (25) are well-known in the CVRP context and it is still an open question whether a polynomial algorithm separating these constraints exists (see for example [ABB+99]). In Subsection (7.2), we explain how to deal with them.

Variables $y_{(i,j)}$ disappear in (RP2). However, for any feasible solution $z_{(i,j)}$ of (RP2), there is a feasible solution to (RP1) with the same $z_{(i,j)}$ (and hence the same value of the objective function), and conversely. This fact is summarized in the following proposition.

Proposition 4. *Let y, z be a feasible solution of (RP1). Then z is a feasible solution of (RP2). Conversely, let z be a solution of (RP2). Then there exists y such that y, z is a feasible solution of (RP1).*

This property is also proven by [HPSG02] when z encodes an Hamiltonian circuit. Here the proposition is proven in a more general case, and the proof is maybe slightly simpler since it does not involve Bender's decomposition but the cut condition for b-flows.

Proof. Take a feasible solution y, z of (RP1). We show that the $z_{(i,j)}$ satisfy (RP2). The only thing that has to be checked is that the $z_{(i,j)}$ satisfy constraints (25). Let $S \subseteq V$. Using constraints (16) and (19), aggregated over $(i,j) \in S$ we get

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{(i,j)} \geq \frac{1}{Q} \sum_{(i,j)\in\delta^+(S)} y_{(i,j)}.$$

Hence

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{(i,j)} \geq \frac{1}{Q} \left(\sum_{(i,j)\in\delta^+(S)} y_{(i,j)} - \sum_{(i,j)\in\delta^-(S)} y_{(i,j)} \right).$$

Since

$$\sum_{(i,j)\in\delta^+(S)} y_{(i,j)} - \sum_{(i,j)\in\delta^-(S)} y_{(i,j)} = \sum_{i\in S} \left(\sum_{j\in V\setminus\{i\}} y_{(i,j)} - \sum_{j\in V\setminus\{i\}} y_{(j,i)} \right),$$

we get with the help of constraints (17) that

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{(i,j)} \geq \frac{1}{Q} d(S).$$

Conversely, take a feasible solution z of (RP2). The only thing that has to be checked is that there exists a non-negative b-flow y with $b_i = p_i - q_i$ for all $i \in V$ with a capacity $= Qz_{(i,j)}$ on each arc (i,j). But constraints (25) are precisely the well-known *cut condition* for flows on networks (see for instance [Gal57]). The integrality is a consequence of the integrality of b_i. □ □

Note that the two programs (RP1) and (RP2) are NP-hard, since they obviously contain the TSP as a special case. Moreover, the continuous relaxation of (RP2) provides a better lower bound than the continuous relaxation of (RP1) thanks to $\lceil\cdot\rceil$ in constraints (25).

6. Relaxation vs original problem

The hardness of SVOCPDP, already emphasized in Subsection 1.1, appears also through the following four propositions, which show that even if we have a feasible solution or an optimal solution of (RP1) or (RP2), it is an NP-complete problem to decide whether this solution is induced by a feasible solution for the SVOCPDP.

Proposition 5. *Let y, z be a feasible solution of (RP1). Deciding whether there is a feasible solution of the* SVOCPDP *inducing y, z is NP-complete.*

Proof. Let $b_1, \ldots, b_r$ be r non-negative integers with $\sum_l b_l$ even. Define $m = \frac{1}{2}\sum_{l=1}^{r} b_l$. Consider then the graph of Figure 8. It encodes a feasible solution y, z of program (RP1): each arc (i,j) has $z_{(i,j)} = 1$ and each number next to it is the corresponding value for $y_{(i,j)}$. Assume that the capacity of the vehicle is $m+1$. We are going to prove now that there is a feasible solution for the SVOCPDP inducing such $y_{(i,j)}$, $z_{(i,j)}$ if and only if there is a subset $I \subseteq \{1, \ldots, r\}$ such that $\sum_{l\in I} b_l = m$, whence showing that deciding whether such a route exists is NP-complete. Note that there are exactly $2m+1$ bikes in the network.

Assume that such a feasible solution exists. Consider again Figure 8 and the following instant: the vehicle takes arc (u, u'). At this time, the vehicle carries $m+1$ bikes. Therefore, it has already taken the arc (s, u) (in order to have m bikes) and exactly one of the arcs (v, v') (in order to get the remaining bike). It cannot have already taken the other arc (v, v'), otherwise it should go back to the depot. It has not taken the arc (u', s) yet. Thus, $m+1$ bikes among the $2m+1$ are on the vehicle, and m bikes are on vertex v': there is no bike left on vertex s. When the vehicle goes back to vertex s, using arc (u', s), it carries exactly m bikes. These m bikes have to be carried to vertex v, using arcs (s, v). Hence, the feasible solution selects (s, v) arcs whose b_l sum exactly to m.

Conversely, assume that there is a subset I such that $\sum_{l\in I} b_l = m$. The sequence starting with the depot, then going through s, u, s in this order, then going back and forth between s and v, using the arcs indexed by I, to carry m bikes to v, then going through v', u, u', s in this order, then using the remaining arcs between

s and v, and finally going through v, v' in order to finish again at the depot is a feasible solution for the SVOCPDP. □ □

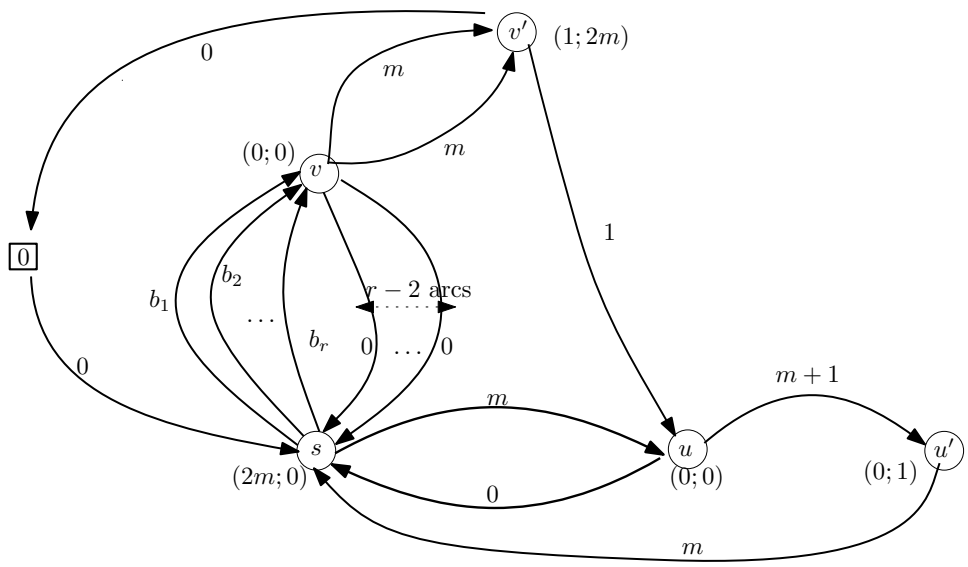

FIGURE 8. Proof of NP-completeness

Proposition 6. *Let z be a feasible solution of (RP2). Deciding whether there is a feasible solution of the* SVOCPDP *inducing z is NP-complete.*

Proof. See Appendix. □ □

Those two propositions deal with feasible solutions of programs (RP1) and (RP2). Now, even if we have a special solution, especially the optimal one, the question whether it is induced by an optimal one of the SVOCPDP is NP-complete. We have indeed the following propositions.

Proposition 7. *Let z^* be an optimal solution of (RP2). Deciding whether there is a feasible solution of the* SVOCPDP *inducing z^* is NP-complete.*

Proof. See Appendix. □ □

Proposition 8. *Let y^*, z^* be an optimal solution of (RP1). Deciding whether there is a feasible solution of the* SVOCPDP *inducing y^*, z^* is NP-complete.*

Proof. See Appendix. □

7. Lower Bound

The integer program (RP2) has an exponential number of capacity and connectivity constraints. The problem is solved through a branch-and-cut algorithm. At each node of the branch-and-cut tree, the continuous relaxation of the problem (RP2) is solved, but only a subset of constraints (24) and (25) are activated. The continuous optimal solution $\bar{z}$ is checked by different routines that try to determinate violated constraints. If a violated constraint is found, it is added into the linear relaxation which is again optimally solved. If no violated constraint is found, but $\bar{z}$ is still fractional, branching goes on.

If the solution is integer, then it is a feasible solution of (RP2). Its value is kept and can be used as an upper bound on the optimal value of (RP2). When the algorithm reaches the end, updating the upper bound during the branch-and-cut process gives at the end the optimal solution of (RP2). If it stops before (for instance if there is a time limit), we get at the end the best feasible solution encountered during its exploration and a lower bound on the optimal solution value, used for calculating the gap.

7.1. **Separation of connectivity constraints.** Given a solution $\bar{z}$, computed at a node of the branch-and-cut tree, we check that constraints (24) are satisfied as follows. We consider the support graph $G[\bar{z}]$ together with a capacity equal to $\bar{z}_{(i,j)}$ on each arc (i,j). For each unbalanced vertex i, we compute the minimum cut $\delta^+(S)$ separating i from the depot 0 (with the algorithm by [EK72]). If this minimum cut has a value strictly lower than 1, then we add the corresponding constraint to the linear program.

7.2. **Separation of capacity constraints.** As already noted when we have defined (RP2), there is no known polynomial algorithm that checks the capacity constraints (25).

The following constraints are less tight but can be separated in polynomial time.

$$\sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} z_{(i,j)} \geq \frac{d(S)}{Q}, \quad (\forall S \subseteq V) \tag{27}$$

These constraints are the relaxation of the constraints (25) in (RP2). They are called "relaxed capacity constraints". A method for finding violated relaxed capacity constraints for the CVRP is proposed by [ABB+99]. In the SVOCPDP, the difference is that a vertex can either be in excess or in default. Therefore their method has been adapted to fit the specific characteristics of the SVOCPDP. The two arcs incident to the vertex 0 are deleted and two new vertices s and t are added. Vertex s is linked to all vertices in excess with the capacity $\kappa_{(s,i)} = \frac{d_i}{Q}$ whereas all vertices in default are linked to vertex t with the capacity $\kappa_{(i,t)} = \frac{-d_i}{Q}$. The capacity is equal to $\bar{z}_{(i,j)}$ on the original arcs (i,j).

We compute an s-t min-cut (again with the Edmonds-Karp algorithm). Let X be the subset of vertices such that $\delta^+(X)$ is the s-t min-cut. s is in X and we define $S := X \setminus \{s\}$. We still define $\bar{S}$ as $V \setminus S$, where V is the vertex set of the original graph, and contains neither s nor t.

$$\text{capacity of } \delta^+(X) = \sum_{i\in S\setminus\{0\}, j\in\bar{S}\setminus\{0\}} \bar{z}_{(i,j)} + \sum_{i\in\bar{S}\setminus\{0\}} \kappa_{(s,i)} + \sum_{i\in S\setminus\{0\}} \kappa_{(i,t)} \tag{28}$$

$$= \sum_{(i,j)\in\delta^+(S)\setminus\delta(0)} \bar{z}_{(i,j)} - \frac{d(S)}{Q} + \sum_{i\in V:\, d(i)>0} \frac{d(i)}{Q}. \tag{29}$$

If the capacity of $\delta^+(X)$ minus $\sum_{i\in V:\, d(i)>0} \frac{d(i)}{Q}$ (which is a fixed value) is less than 0, it means that a relaxed capacity constraint is violated and the corresponding constraint are added to the linear relaxation. Otherwise, there is no violated relaxed capacity constraint as shown in Figure 9, which represents a possible solution of the linear relaxation when solving the instance presented in Figure 2, but with a capacity of the vehicle set to $Q = 10$. The fractional value of each arc is given next to it.

When the relaxed capacity constraints are respected a second procedure runs looking for violated capacity constraints. The interest to look for these constraints in addition to the former is to increase the speed of the general algorithm. Indeed, we get in this case tighter constraints. The used procedure is again an heuristic presented in [ABB+99], since there is no known polynomial algorithm. It is a tabu search that tries to find a subset S violating the constraints. A short explanation of the tabu search method is done later in Section 8. The main idea is starting from a subset S of vertices, vertices are either added to S or removed from S in order to find a new subset where capacity constraints are violated. Roughly speaking, the vertex that is added to or removed from the subset is kept in a tabu list and cannot be used for a given number of iterations. If the procedure finds a violated capacity constraint, it is added to the linear program.

7.3. **Initial relaxation, separation strategy and branching rules.** The initial relaxation solved is (RP2) without (24) and with (25) only for $S = \{i\}$ for all $i \in V$. The separation routines are called with respect to the following order:

- connectivity constraints (24)
- relaxed capacity constraints (25)
- capacity constraints (27)

The branch-and-bound tree framework SCIP has been used for the computational results. Two different branching strategies have been tested.

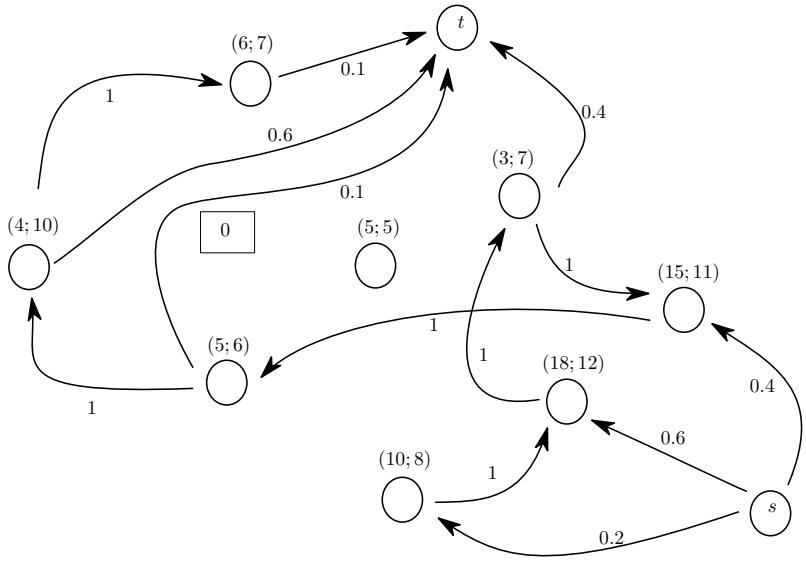

FIGURE 9. Construction of the new graph from the graph of Figure 2 to check the relaxed capacity constraints

The first one is the so-called *reliability branching rule* defined in their paper (see in [AKM05] for more details), which is the one that seems the most effective one for solving SVOCPDP after several tests that have been done. For the node selector, the rule is the *best estimate search rule*, which is the default rule of SCIP.

The second one consists in branching first on $\sum_{j \in V} z_{(j,i)}$. If there is a vertex i for which this quantity is equal to a non-integral u, we add two constraints:

$$\sum_{j \in V} z_{(j,i)} \geq \lceil u \rceil \text{ and } \sum_{j \in V} z_{(j,i)} \leq \lfloor u \rfloor.$$

The vertex for which $u - \lfloor u \rfloor$ is the nearest from 0.5 is selected in priority. When there is no such vertex, the branching uses the former strategy. This second strategy (called in the following *degree branching rule*) is interesting since it based on the idea of branching on constraints instead of branching simply on variables. For the node selector, the rule is the same of the previous branching.

7.4. **Adding valid inequalities.** The clique cluster inequalities defined in Section 3.1 of [HPSG07] can be straightforwardly adaptated. We have implemented them but the fact that the variables are not restricted to binary values have lead to huge computation times. It comes partially from the fact that their efficient version (7) of that paper is no longer valid. The still valid version is separated in a more difficult way. The results were dominated by the version without them. We have not included the results in the present paper. It seems that adding cuts in efficient way for the SVOCPDP is an open problem. Other inequalities could be adaptated, such as the comb inequalities. We have not tried them.

8. Upper Bound

The algorithm chosen for computing an upper bound of the optimal value is a tabu search. Several problems like: job shop scheduling, graph coloring, traveling salesman problem and other vehicle routing have been successfully tackled by the tabu search. For an introduction to this method, see for instance [GL97]. While a greedy local search stops when no improving move is found in the neighborhood $N(s)$ of the current solution s, the basic principle of a tabu search is to continue the search by allowing non-improving moves. To avoid cycling on the same solution, the last ℓ moves are kept in the *tabu list* that is updated at each iteration. Its size ℓ is a parameter.

Various stopping criteria can be used, such for example: the predetermined number of iterations with no improvement, the maximum number of iterations, the maximum amount of time spent in the method. To define a tabu search properly, we have to describe how to compute the cost of the current solution, the neighborhood, the way to encode the tabu list, a heuristic to build the first current solution and the stopping criterion.

8.1. **Cost of the current solution.** Proposition 1 allows to encode the solution as a sequence of vertices, starting and ending with the depot 0, without specifying bike displacements. The score of a sequence $i_1 \to i_2 \ldots \to i_k$ is the cost of the traveled distance $\sum_{j=1}^{k-1} c_{(i_j,i_{j+1})}$ to which we add a penalty when the sequence is infeasible, i.e. when there is no feasible solution for the SVOCPDP inducing this sequence. The feasible sequence space may be disconnected, whence to explore efficiently the used neighborhood is enlarged allowing the tabu in visiting infeasible solutions. Then, the infeasibility is penalized in the objective function. Thanks to Proposition 1 we are able to evaluate the infeasibility of any sequence indicated in the following with s. It is the "degree of infeasibility" we have defined in Section 3. The relaxed constraints are (8) and (9) and the resulting cost function $f(s)$ defined in (30) is inspired by the one proposed by [GHL94] for the VRP and is the following:

$$f(s) = \sum_{j=1}^{k-1} c_{(i_j,i_{j+1})} + \gamma \sum_{i \in V} (p_i - p'_i) \tag{30}$$

where γ is a positive constant and the p'_i are computed through the max-flow algorithm of Proposition 1. Recall that according to that proposition, $p'_i \leq p_i$.

In our experiment we have defined γ as 10 times the mean distance of a direct trip from the depot to a vertex. This choice is a way to convert at the right scale in term of cost the number of bikes that remain to be displaced (and was experimentally tuned).

8.2. **Initial solution.** We propose two distinct methods to compute the initial sequence needed for the tabu search.

8.2.1. *Greedy heuristics.* The method first tries to *close* vertices. Closing a vertex means bringing a vertex from its original state to its target state q_i in one move. The first criterion used, for deciding which vertex visit first, consists in ranking the unbalanced vertices with respect to their distance from the vehicle position and then in choosing the nearest vertex it can close. If it is impossible to close any vertex, then for each vertex the number of bikes the vehicle can load or unload is computed and the vehicle is driven toward the vertex with the highest absolute number of bikes that can be exchanged (i.e., either loaded or unloaded). In case of equality between several vertices, it goes to the nearest one. With this method we are sure to start from a feasible sequence.

8.2.2. *The solution of (RP2).* A solution z of relaxation (RP2) of Section 5 is such that $G[z]$ is an Eulerian graph. A closed Eulerian path in the supported graph $G[z]$ provides a sequence of vertices that can be used as an initial solution of the tabu search. To compute such a closed Eulerian path various classical methods are available (see for instance page 31 of the book by [KV02]). Since the relaxation is quite good, this sequence is often not too far from an optimal solution. Note that if the sequence is feasible for the SVOCPDP, it is an optimal solution. Unfortunately, even if we have an instance for which the relaxation is tight, there is no polynomial algorithm for finding the optimal solution of the original problem (Proposition 7 above, Section 6).

8.3. **Neighborhood description.** At each iteration the whole neighborhood is explored. The definition of neighborhood is essential, since different neighborhoods offer different ways to explore the solution space but require different computational efforts. For solving the addressed problem, four different moves have been defined and used in the tabu search framework. Note that any time a vertex appears two consecutive times in a sequence one of the occurrences can be removed.

Let k denote the length of the sequence and note that k is in general larger than n. We therefore evaluate the complexity for computing the neighborhood in terms of k instead of n. To illustrate the different moves the solution displayed on Figure 2 is used each time as initial solution.

$$0 \to 7 \to 6 \to 4 \to 5 \to 2 \to 8 \to 1 \to 0$$

Following this sequence, the vehicle leaves the depot to go to vertex 7. Then it goes on with vertex 6 and so on until vertex 1 from where it goes back to the depot.

8.3.1. *2-OPT.* This move is classical for routing problems. A pair of non consecutive arcs are removed from the sequence and two new arcs are inserted. They link the two tails and the two heads of the removed arcs and therefore the travelling direction of the subset of nodes between the two removed arcs is reversed to obtain a new sequence. An example of 2-OPT is obtained by removing arcs (0,7) and (5,2), by inserting arcs (7,2) and (0,5) and reversing the sequence $5, 4, 6, 7$, therefore the following sequence is obtained:

$$0 \to 5 \to 4 \to 6 \to 7 \to 2 \to 8 \to 1 \to 0$$

This move is tested for each pair of arcs in the sequence. It takes $O(k^2)$ to test all the moves of this type.

8.3.2. *Suppression.* This move tries to delete a vertex in the sequence. If we delete vertex 4 in the example of Figure 2, we obtain

$$0 \to 7 \to 6 \to 5 \to 2 \to 8 \to 1 \to 0$$

This move is tested for each vertex in the sequence. It takes $O(k)$ to test all moves of this type.

8.3.3. *Add unbalanced vertex.* This move is active when the current sequence is infeasible. All the vertices are checked and both the most unbalanced vertex i in excess and the most unbalanced vertex j in default are selected (vertices for which $|(q_i - q'_i) - (p_i - p'_i)|$ are maximal). One neighbor is obtained by adding, just after i in the sequence, the moves $\to j \to i \to$. The other neighbor is obtained by adding the moves $\to i \to j \to$ in the sequence just after j. If neither i nor j are present in the sequence, then there is only the neighbor obtained by adding $\to i \to j$ at the end of the sequence. Testing all moves of this type takes $O(k^2)$.

Once more with the example reported in Figure 2, if the capacity of the vehicle is lower than 8, the vertices are not balanced at the end of the sequence. Let $Q = 6$, the vehicle cannot bring vertices 1, 6 and 8 to their target state: 2 bikes need to be loaded from 6 while 1 bike is requested by both node 1 and node 8. The moves $\to 8 \to 6$ are tried to be added and we obtain the following sequence

$$0 \to 7 \to 6 \to 8 \to 6 \to 4 \to 5 \to 2 \to 8 \to 1 \to 0$$

This sequence is a feasible solution.

8.3.4. *Add buffer vertex.* This move tries to add a second time a vertex in the sequence. The second copy of a vertex can be used as buffer vertex, since in the buffer vertex bikes are either delivered or picked up. Thus, if a vertex is used as a buffer vertex, it has to be visited at least twice. Every vertex can be used as a buffer vertex. A copy of each vertex already present in the sequence is tried to be inserted at each possible position. If a vertex is not present in the sequence (for example the initially balanced vertices), then it is inserted twice.

This move is very time consuming and testing all moves of this type takes $O(k^3)$. Thus it is not called at each iteration but in 20% of the cases.

8.4. **The tabu list.** The fact that arcs could be used more than once leads to a management of the tabu list slightly different from the classical one used, for example, for solving classical routing problems. When a move is made, some arcs leave the solution and others enter the solution. Each exiting arc is kept in the tabu list along with the position of its occurrence in the last solution. This arc is forbidden for ℓ iterations, but it is allowed to be reinserted in a different position.

The tabu list is kept using the `multimap` structure in the STL of C++, which ensures a quick access.

8.5. **The tabu search.** The components previously described are integrated in the general algorithm summarized in Algorithm 1. The following notations are used in the pseudo-code: s is the current solution; s^* is the best feasible solution encountered from the beginning of the tabu search. $f(s)$ is the value of the objective function for the solution s; $NbIterMax$ is the maximum number of iterations done before the tabu search stops. i is the current iteration; $\bar{s}_X$ is the best solution encountered using the move X; $s^{\#}$ is the best solution in the whole neighborhood. a is a random variable uniformly distributed between 0 and 1.

Algorithm 1 Tabu search algorithm

```
s ← ComputeInitialSolution()
s* ← s
i ← 0
while i ≤ NbIterMax do
   s̄_2OPT ← Explore2OPT(s)
   s̄_Sup← ExploreSuppresion(s)
   s# ← argmin(f(s̄_2OPT), f(s̄_Sup))
   if s is not a route then
      s̄_AddUnb ← ExploreAddUnbalanced(s)
      if f(s#) > f(s̄_AddUnb) then
         s# ← s̄_AddUnb
      end if
   end if
   a ← Random()
   if a ≤ 0.2 then
      s̄_AddBuf ← ExploreAddBuffer(s)
      if f(s#) > f(s̄_AddBuf) then
         s# ← s̄_AddBuf
      end if
   end if
   s ← s# and Update the tabu list
   if s is a route and f(s) < f(s*) then
      s* ← s
   end if
   i ← i + 1
end while
```

9. Computational Study

The algorithms have been coded in C++, embedded into SCIP (a Constraint Integer Programming framework, see [AKM05]) and tested on a PC AMD Athlon 5600+ clocked at 2.8 GHz, with 16 GB RAM. To the best of our knowledge, this paper is the first solving instances of the SVOCPDP, therefore the following Subsection 9.1 presents how the instances have been created, the results obtained are reported in Subsection 9.2 while Subsection 9.3 reports a discussion on the results.

9.1. **Instances.** The instances are taken from [HPSG04a] which defines, for various values of n, 10 instances named from A to J, with an algebraic demand $\tilde{d}_i$ between -10 and 10 for each vertex i. We work on these instances for $n \in \{20, 40, 60, 100\}$. In order to get an initial state p_i and a target state q_i for each vertex i, as required for the SVOCPDP, the demands have been modified. We do it twofold. A first set of instances has been obtained by setting $p_i = 10$, $q_i = 10 + \tilde{d}_i$ and $C_i = 20$ for each vertex i. We refer to these instances by the indication $\bar{p}_i = \bar{q}_i = 10$ (average state). A second set of instances is obtained by setting $p_i = 30$, $q_i = 30 + 3 \times \tilde{d}_i$ and $C_i = 60$ for each vertex i. We refer to these instances by the indication $\bar{p}_i = \bar{q}_i = 30$ (average state).

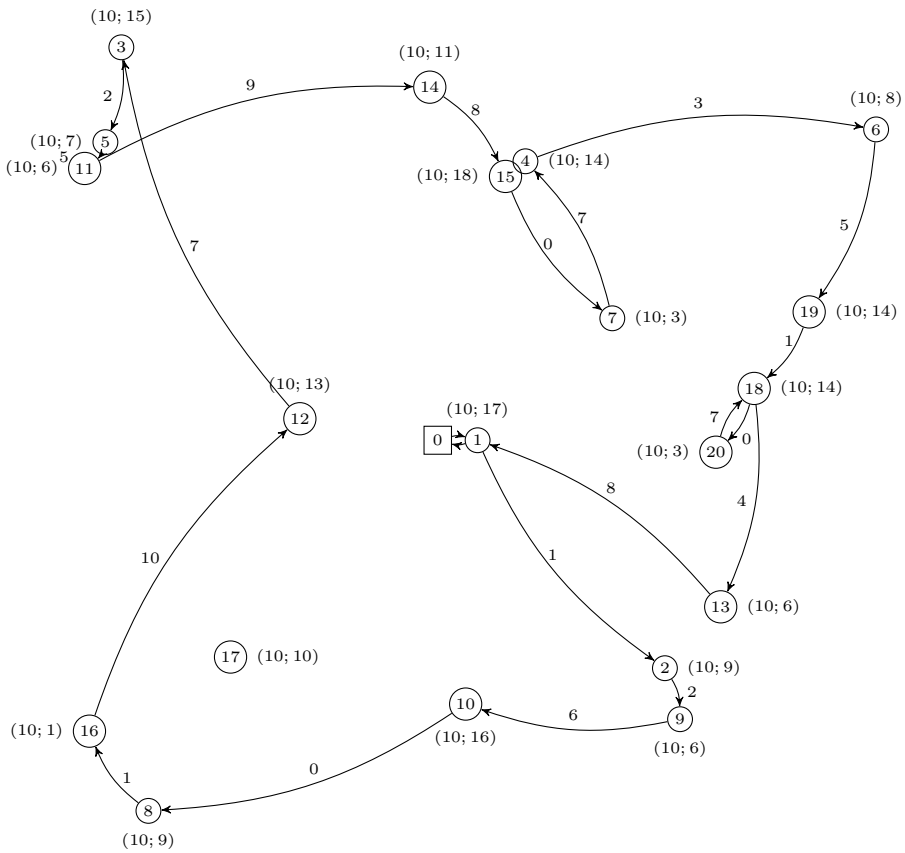

FIGURE 10. An optimal solution already found by the branch-and-cut for the instance n20G, with $Q = 10$ and $\bar{p}_i = \bar{q}_i = 10$

9.2. **Computational results.** The proposed algorithms have been tested on these instances for all $Q \in \{10, 30, 45, 1000\}$. Note that even with huge capacity for the vehicle our problem is not a TSP with pickup and delivery. Initially balanced vertices can be skipped, nevertheless visiting a vertex several times can improve the cost of the solution (see for example the solution reported in Figures 4 and 5). The algorithms tested are the tabu search initialized with the greedy heuristics from Subsection 8.2.1, the branch-and-cut of Section 7 which provides a lower bound for the problem, and the tabu search initialized with the solution of the branch-and-cut as explained in Subsection 8.2.2.

The results achieved by all our algorithms for solving the generated instances are reported respectively in Tables 1, 3, 5 and 7 for the reliability branching rule and the tabu search, and Tables 9, 11, 13 and 15 for the degree branching rule. For each pair n, Q, we give the results for two instances – the one achieving the smallest gap and the one achieving the largest one – and also the average performance on the ten instances. We have treated the results for $n = 100$ in distinct tables since the results obtained for such dimensions are less good than those obtained with $n < 100$. The following webpage gathers the best results we have obtained so far for the complete set of instances.

http://cermics.enpc.fr/∼meuniefr/SVOCPDP.html

The tabu search initialized with the greedy heuristic (indicated in the following with TS1) has the following set of parameters: maximum number of iterations = $1'000$, the size of the tabu list (i.e. ℓ) = 30 and the maximum amount of time = $1'000$ seconds. The method also stops when no improvement has been done for 80 consecutive iterations. The tabu search initialized with the solution obtained by the branch-and-cut

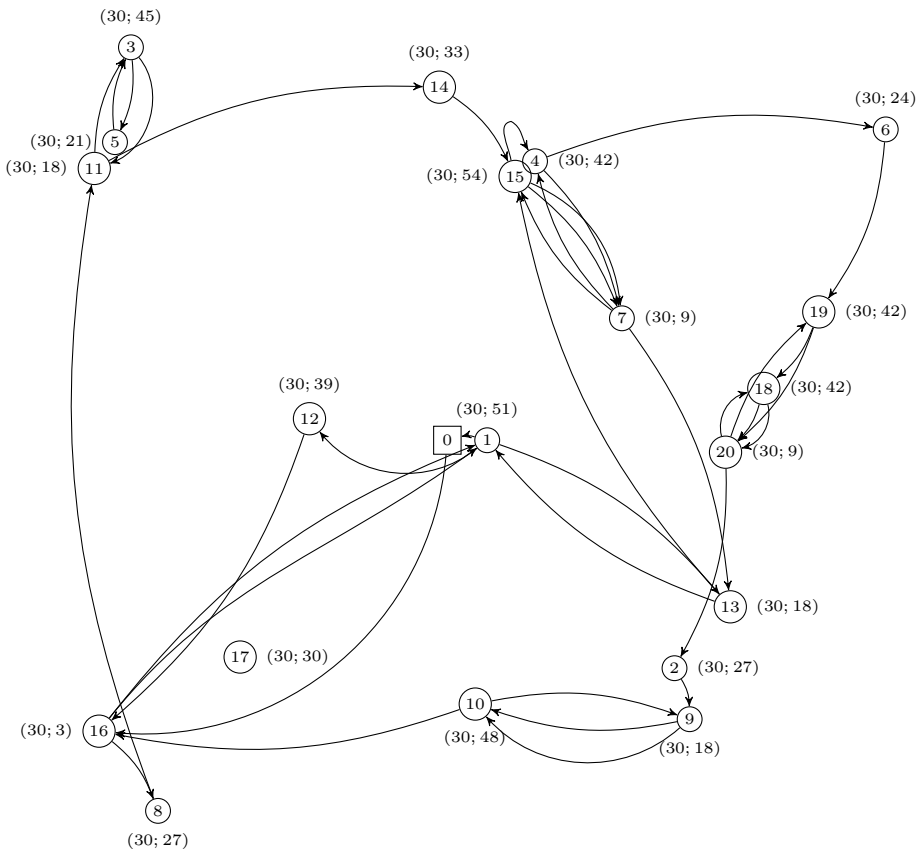

FIGURE 11. An optimal solution already found by the branch-and-cut for the instance n20G, with $Q = 10$ and $\bar{p}_i = \bar{q}_i = 30$

(indicated in the following with TS2) has the following parameters: the tabu list size is set to 3, the maximum number of iterations to 50 and the maximum amount of time to 300 seconds. The search is stopped whenever 15 consecutive iterations did not improve the best solution or if a solution is found with a cost equal to the lower bound obtained by the branch-and-cut. The abbreviations used in these tables are the following:

n is the number of vertices.
Q is the vehicle capacity.
UB1 is the value obtained by TS1.
Time is the cpu time (in seconds) used by the Tabu Search TS1.
Iter. is the number of iterations.
UB2 is the best solution obtained by the tabu searches TS1 and TS2.
T.t. is the cpu time (in seconds) elapsed for executing TS1+B&C+TS2.
LB is the best lower bound found by the branch-and-cut on (RP2).
Gap % is the percentage gap between the previous LB and UB2.

The results achieved by the branch-and-cut alone are reported respectively in Tables 2, 4, 6 and 8 for the reliability branching rule, and Tables 10, 12, 14 and 16 for the degree branching rule. The maximum time was set to $10'000$ seconds. The abbreviations used in these tables are the following:

n is the number of vertices.
Q is the vehicle capacity.
UB-BC is the best solution found by the branch-and-cut.

LB is a lower bound on the optimal solution of the relaxation.
LBr is the lower bound that has been found at the root node.
Gap % is the percentage gap between UB-BC and LB.
Time is the computational time.
 # Nodes is the number of nodes of the Branch-and-Cut tree.

TABLE 1. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$, less than 60 vertices, and a vertex capacity equal to 20 – **reliability branching**

Instance name	n	Q	UB1	Time	Iter.	UB2	T.t.	LB	Gap %
n20A	20	10	4764	9	132	4702	10	4702.00	0.00
n20B	20	10	4776	7	124	4769	8	4769.00	0.00
	20	10		7	126		9		0.00
n20D	20	30	4156	4	93	4089	5	4089.00	0.00
n20E	20	30	4556	5	88	4556	5	4299.00	5.97
	20	30		5	105		6		1.13
n20B	20	45	3792	5	102	3792	5	3792.00	0.00
n20C	20	45	4045	8	88	4045	8	3891.00	3.96
	20	45		6	129		7		0.83
n20B	20	1000	3792	7	90	3792	7	3792.00	0.00
n20C	20	1000	4042	6	138	4042	6	3891.00	3.88
	20	1000		7	138		8		0.83
n40B	40	10	6377	198	121	5949	468	5949.00	0.00
n40F	40	10	7373	391	216	7138	10473	6651.57	7.31
	40	10		367	220		4266		1.54
n40B	40	30	5297	142	91	5110	145	5110.00	0.00
n40C	40	30	4806	261	179	4692	262	4644.00	1.03
	40	30		225	157		228		0.10
n40B	40	45	5297	141	136	5110	144	5110.00	0.00
n40C	40	45	4941	185	113	4656	187	4522.00	2.96
	40	45		227	110		230		0.53
n40B	40	1000	5297	152	91	5110	157	5110.00	0.00
n40H	40	1000	5060	183	128	5006	187	4772.00	4.90
	40	1000		273	130		279		0.96
n60H	60	10	9100	1007	76	8123	11043	7747.60	4.84
n60F	60	10	9234	1004	54	8374	11034	7387.74	13.35
	60	10		1003	69		11036		10.48
n60J	60	30	6543	1026	85	6389	1177	6389.00	0.00
n60I	60	30	6918	995	77	6480	11028	6153.33	5.31
	60	30		1013	73		2218		1.05
n60J	60	45	6602	988	60	6374	1047	6374.00	0.00
n60H	60	45	6122	1037	51	6122	1055	5825.00	5.09
	60	45		998	67		1046		2.25
n60D	60	1000	6345	1022	57	6223	1946	6223.00	0.00
n60I	60	1000	6387	988	65	6355	1469	5914.00	7.45
	60	1000		1007	67		1255		2.19

Figure 10 gives the solution for the instance n20G with $Q = 10$, and an average number of bikes per station equal to 10. The depot appears as a square with a 0 and $(p_i; q_i)$ is given next to each vertex. The solution displayed is the one obtained at the end of the following sequence of algorithms: the tabu search first, then the branch-and-cut with the result of the tabu search as upper bound, and, finally, the tabu search which starts from the solution given by the branch-and-cut. In this example the route displayed is exactly the solution of the branch-and-cut. It is the optimal solution even if a vertex (vertex 18) is visited twice. Compare with Figure 11. It is the same instance, but this time with an average number of bikes per station equal to 30. Again, the branch-and-cut is able to find the optimal solution. Interestingly, we can see that, with an increasing average number of bikes but with the same vehicle capacity, optimal solutions may visit vertices several times.

9.3. **Discussion of the performances.** As expected, the computational time and the number of nodes of the branch-and-cut algorithm increase with the number of vertices and decrease with the capacity of the vehicle. In Tables 2, 4, 6 and 8 (and 10, 12, 14 and 16), the number of nodes used in the branch-and-cut could seem really huge. However, since variables are integers (not binary), several branchings must be done on the same variable before obtaining the optimal solution. Except for the large instances with 100 vertices, the results reported in Tables 1 and 5 show that the distance between the best upper bound found and the lower bound is really small. The gap is on average less that 5%. The local search is very efficient on small and medium instances (up to 60 vertices) and becomes less effective for larger instances. This is probably a

TABLE 2. Performance of the branch-and-cut cited in Table 1 – **reliability branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n20A	20	10	4702	4702.00	4634.23	0.00	0.87	23
n20B	20	10	4769	4769.00	4683.50	0.00	0.91	87
	20	10				0.00	1.77	350
n20D	20	30	4089	4089.00	4089.00	0.00	0.19	1
n20E	20	30	4299	4299.00	4299.00	0.00	0.20	1
	20	30				0.00	0.18	3
n20B	20	45	3792	3792.00	3792.00	0.00	0.07	1
n20C	20	45	3891	3891.00	3891.00	0.00	0.06	1
	20	45				0.00	0.15	6
n20B	20	1000	3792	3792.00	3792.00	0.00	0.20	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.09	1
	20	1000				0.00	0.19	2
n40B	40	10	5949	5949.00	5723.67	0.00	45.07	2507
n40F	40	10	7147	6651.57	6371.78	7.45	10000.00	715318
	40	10				1.56	3863.36	274129
n40B	40	30	5110	5110.00	5086.50	0.00	2.62	9
n40C	40	30	4644	4644.00	4644.00	0.00	1.17	1
	40	30				0.00	2.46	28
n40B	40	45	5110	5110.00	5040.50	0.00	2.37	26
n40C	40	45	4522	4522.00	4522.00	0.00	1.16	1
	40	45				0.00	2.57	33
n40B	40	1000	5110	5110.00	5040.50	0.00	4.75	29
n40H	40	1000	4772	4772.00	4765.50	0.00	3.17	9
	40	1000				0.00	4.56	18
n60H	60	10	8147	7547.60	7664.40	5.15	10000.00	157357
n60F	60	10	8555	7387.74	7257.74	15.80	10000.00	113536
	60	10				15.20	10000.00	148362
n60J	60	30	6389	6389.00	6151.83	0.00	149.12	2598
n60I	60	30	6390	6153.33	6021.00	3.84	10000.00	245505
	60	30				0.38	1199.32	28002
n60J	60	45	6374	6374.00	5974.07	0.00	129.90	1701
n60H	60	45	5825	5825.00	5751.17	0.00	17.68	110
	60	45				0.00	47.50	612
n60D	60	1000	6223	6223.00	6055.29	0.00	915.27	1752
n60I	60	1000	5914	5914.00	5729.00	0.00	474.56	1627
	60	1000				0.00	244.17	613

TABLE 3. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$ and 100 vertices of capacity 20 – **reliability branching**

Instance name	**n**	**Q**	**UB1**	**Time**	**Iter.**	**UB2**	**T.t.**	**LB**	**Gap %**
n100F	100	10	14759	1184	5	12621	11206	10048.40	25.60
n100I	100	10	17799	1181	7	16602	11206	11829.80	40.34
	100	10		1176	4		11210		32.54
n100A	100	30	12175	1262	4	8033	11282	7496.55	7.15
n100B	100	30	11066	1309	5	10223	11337	7441.96	37.37
	100	30		1066	4		11191		16.60
n100H	100	45	9506	1172	4	7796	11191	7548.23	3.28
n100B	100	45	8838	954	11	8020	10975	6941.07	15.54
	100	45		1317	6		11340		9.68
n100H	100	1000	9275	968	2	7786	8215	7549.00	3.14
n100J	100	1000	9178	1087	1	8655	11114	7045.60	22.84
	100	1000		1187	3		10934		12.92

TABLE 4. Performance of the branch-and-cut cited in Table 3 – **reliability branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n100F	100	10	13022	10048.37	10036.87	29.59	10000.03	29066
n100I	100	10	17069	11829.79	11771.75	44.28	10000.12	23018
	100	10				40.07	10000.05	25279
n100A	100	30	8127	7496.55	7386.66	8.40	10000.01	36791
n100B	100	30	9445	7441.96	7378.37	26.91	10000.04	35819
	100	30				16.87	10000.03	37999
n100H	100	45	7742	7548.22	7513.67	2.56	10000.00	36301
n100B	100	45	8177	6941.06	6869.58	17.80	10000.09	30280
	100	45				9.33	9246.13	33951
n100H	100	1000	7549	7549.00	7522.22	0.00	7225.74	936
n100J	100	1000	8868	7045.60	7039.61	25.86	10001.42	865
	100	1000				13.34	9724.77	1081

TABLE 5. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$, less than 60 vertices and a vertex capacity equal to 60 – **reliability branching**

Instance name	**n**	**Q**	**UB1**	**Time**	**Iter.**	**UB2**	**T.t.**	**LB**	**Gap %**
n20A	20	10	9112	220	1453	9084	122	9084.00	0.00
n20B	20	10	9888	120	1453	9883	56	9883.00	0.00
	20	10		176	1		126		0.00
n20A	20	30	5010	6	97	4702	8	4702.00	0.00
n20B	20	30	5009	6	107	4769	7	4769.00	0.00
	20	30		8	153		11		0.00
n20B	20	45	4174	5	99	4174	5	4174.00	0.00
n20C	20	45	5295	5	90	5137	6	5113.00	0.47
	20	45		8	124		11		0.05
n20B	20	1000	3792	5	100	3792	5	3792.00	0.00
n20C	20	1000	4042	8	143	4042	8	3891.00	3.88
	20	1000		6	125		7		0.84
n40E	40	10	13527	994	69	13159	5286	13159.00	0.00
n40F	40	10	15447	1009	71	15439	11078	14534.00	6.22
	40	10		993	59		9697		2.38
n40E	40	30	6645	312	202	6424	790	6424.00	0.00
n40F	40	30	7576	284	155	7074	10351	6709.18	5.44
	40	30		247	159		3710		1.04
n40E	40	45	5951	137	95	5671	474	5671.00	0.00
n40C	40	45	6252	618	116	5847	1059	5847.00	0.00
	40	45		255	184		560		0.00
n40B	40	1000	5297	127	91	5110	132	5110.00	0.00
n40C	40	1000	4814	195	135	4656	197	4552.00	2.96
	40	1000		137	5		233		0.49
n60H	60	10	27776	774	1	17100	10813	16230.10	5.36
n60C	60	10	28405	759	3	20918	10799	17799.60	17.51
	60	10		978	2		11012		10.52
n60H	60	30	9562	1030	86	8216	11062	7672.27	7.09
n60C	60	30	10355	1031	78	9667	11060	8360.76	15.62
	60	30		1004	73		11035		12.81
n60H	60	45	7333	992	78	6743	2765	6743.00	0.00
n60D	60	45	9327	1010	106	8793	11049	7631.00	15.22
	60	45		1003	82		9349		6.62
n60J	60	1000	6598	986	58	6374	1237	6374.00	0.00
n60I	60	1000	6376	1022	78	6355	1610	5914.00	7.45
	60	1000		1010	74		1177		2.39

consequence of the size of the neighborhood, which can be quite huge when the vehicle has to make several visits at some vertices. For $n = 100$, the second tabu search makes sometimes only one or two iterations.

Note also that the smaller is the capacity, the harder is the problem. It is in accordance with the intuition, since for instances with small vehicle capacity, the mean number of visits by vertex increases. Instances n20B for $Q = 10$ and $Q = 45$ and n40E for $Q = 10$ and $Q = 30$ in Table 1 are illustrations for such a phenomenon. We have also such a phenomenon for n20B and n40E in Table 5 for $Q = 10$ and $Q = 30$.

The reliability branching rule seems to be slightly better than the degree branching one. However, this latter improves sometimes the lower bound: for instance, n60I, with $Q = 30$ and 10 bikes per station in average (Tables 1 and 9), or n60D, with $Q = 45$ and 30 bikes per station in average (Tables 5 and 13).

If we go back to the size of the *Vélib* system in Paris (one truck of capacity 20 for 50 vertices of capacity 30), the instances that are close to these numerical features are the instances with $n = 60$, $Q = 30$, and 10 bikes in average per station (Tables 1 and 9) and the instances with $n = 40$, $Q = 30$ and 30 bikes in average per station (Tables 5 and 13). We get solutions that have most of the time a gap smaller than 2%. It would certainly be possible to improve the time consumption of the flow algorithm in the tabu search. Stopping the branch-and-cut after a fixed time would also be a reasonable solution to reduce the total time to get a good solution. However, we see that the instances of this kind can already be solved within an average optimality gap of less than 2% in less than 40 minutes.

REFERENCES

[ABB+99] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberan, and D. Naddef, *Separating capacity constraints in the CVRP using tabu search*, European Journal of Operational Research **106** (1999), 546–557.

[AH92] S. Anily and R. Hassin, *The swapping problem*, Networks **22** (1992), 419 – 433.

TABLE 6. Performance of the branch-and-cut cited in Table 5 – **reliability branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n20A	20	10	9084	9084.00	9074.13	0.00	0.35	5
n20B	20	10	9883	9883.00	9872.33	0.00	0.32	9
	20	10				0.00	25.84	13522
n20A	20	30	4702	4702.00	4629.67	0.00	1.64	211
n20B	20	30	4769	4769.00	4749.75	0.00	0.28	9
	20	30				0.00	2.48	704
n20B	20	45	4174	4174.00	4174.00	0.00	0.12	1
n20C	20	45	5113	5113.00	5041.50	0.00	0.30	19
	20	45				0.00	2.35	715
n20B	20	1000	3792	3792.00	3792.00	0.00	0.14	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.09	1
	20	1000				0.00	1.18	2
n40E	40	10	13159	13159.00	12714.22	0.00	4261.69	269378
n40F	40	10	15447	14534.03	14263.84	6.28	10000.00	663752
	40	10				2.49	8649.17	501582
n40E	40	30	6424	6424.00	6126.93	0.00	473.11	27603
n40F	40	30	7074	6709.18	6477.35	5.43	10000.00	597725
	40	30				1.04	3439.25	221229
n40E	40	45	5671	5671.00	5247.88	0.00	333.72	19408
n40C	40	45	5912	5912.00	5513.40	0.00	378.41	25286
	40	45				0.00	302.35	20201
n40B	40	1000	5110	5110.00	5040.50	0.00	4.49	25
n40C	40	1000	4522	4522.00	4522.00	0.00	1.93	1
	40	1000				0.00	10.68	88
n60H	60	10	17601	16230.13	16119.97	8.44	10000.35	144438
n60C	60	10	21393	17799.60	17684.76	20.18	10000.32	159183
	60	10				14.41	10000.00	153241
n60H	60	30	8379	7672.26	7544.15	9.21	10000.01	153746
n60C	60	30	9933	8360.76	8215.37	18.80	10000.01	176011
	60	30				17.14	10000.01	150086
n60H	60	45	6743	6743.00	6451.97	0.00	1769.15	30694
n60D	60	45	8928	7631.00	7492.50	16.99	10000.30	153945
	60	45				8.54	8313.48	156158
n60J	60	1000	6374	6374.00	6214.46	0.00	247.26	643
n60I	60	1000	5914	5914.00	5729.00	0.00	581.50	1885
	60	1000				0.00	165.18 422	

TABLE 7. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$ and 100 vertices of capacity 60 – **reliability branching**

Instance name	**n**	**Q**	**UB1**	**Time**	**Iter.**	**UB2**	**T.t.**	**LB**	**Gap %**
n100D	100	10	47489	18910	1	37972	29235	31412.70	20.88
n100E	100	10	41002	1889	1	30222	12021	23311.00	29.65
	100	10		6989	1		17146		25.50
n100A	100	30	16110	1075	2	13366	11101	10345.00	29.20
n100B	100	30	18739	844	1	15537	10866	11114.10	39.80
	100	30		1076	4		11118		34.70
n100A	100	45	11372	1747	11	10694	11766	8335.32	28.30
n100D	100	45	15845	1034	2	15845	11053	9745.95	62.58
	100	45		1238	6		11266		39.98
n100D	100	1000	9832	1294	3	7595	11318	7352.00	3.30
n100F	100	1000	9371	1756	10	8783	11777	7288.92	20.50
	100	1000		1176	4		10282		10.65

TABLE 8. Performance of the branch-and-cut cited in Table 7 – **reliability branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n100D	100	10	40057	31412.72	31375.48	27.52	10000.03	13188
n100E	100	10	31898	23311.01	23293.71	36.84	10000.00	18166
	100	10				33.11	10000.05	16045
n100A	100	30	14228	10345.00	10339.62	37.54	10000.01	26628
n100B	100	30	16059	11114.08	11088.10	44.49	10000.04	24334
	100	30				41.82	10000.43	24077
n100A	100	45	10647	8335.32	8271.49	27.73	10000.03	34072
n100D	100	45	13762	9745.95	9695.37	41.21	10000.01	25337
	100	45				34.38	10000.03	33923
n100D	100	1000	7593	7352.00	7347.37	3.28	10003.89	933
n100F	100	1000	9017	7288.92	7273.37	23.71	10000.23	1146
	100	1000				11.37	9086.88	3811

TABLE 9. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$, less than 60 vertices, and a vertex capacity equal to 20 – **degree branching**

Instance name	**n**	**Q**	**UB2**	**T.t.**	**LB**	**Gap %**
n20A	20	10	4702	8	4702.00	0.00
n20B	20	10	4769	6	4769.00	0.00
	20	10		13		0.00
n20D	20	30	4089	14	4089.00	0.00
n20E	20	30	4574	6	4299.00	6.40
	20	30		7		1.13
n20B	20	45	3792	5	3792.00	0.00
n20C	20	45	4042	8	3891.00	3.88
	20	45		7		1.13
n20B	20	1000	3792	6	3792.00	0.00
n20C	20	1000	4045	6	3891.00	3.96
	20	1000		7		0.83
n40B	40	10	5949	468	5949.00	0.00
n40A	40	10	7063	10363	6418.81	10.04
	40	10		9344		5.63
n40B	40	30	5110	181	5110.00	0.00
n40C	40	30	4822	297	4644.00	3.83
	40	30		222		0.38
n40B	40	45	5110	221	5110.00	0.00
n40C	40	45	4656	192	4522.00	2.96
	40	45		223		0.49
n40B	40	1000	5110	173	5110.00	0.00
n40C	40	1000	4656	311	4522.00	2.96
	40	1000		232		0.67
n60H	60	10	8129	11025	7592.00	7.07
n60D	60	10	11530	11044	9377.78	22.95
	60	10		11049		15.32
n60J	60	30	6389	1054	6389.00	0.00
n60I	60	30	6480	11028	6209.50	4.36
	60	30		2590		0.95
n60J	60	45	6374	1047	6374.00	0.00
n60A	60	45	5945	1088	5740.00	3.57
	60	45		1086		1.59
n60D	60	1000	6223	1215	6223.00	0.00
n60F	60	1000	6109	1182	5778.00	5.73
	60	1000		1258		1.93

[AKM05] T. Achterberg, T. Koch, and A. Martin, *Branching rules revisited*, Operations Research Letters **33** (2005), no. 1, 42 – 54.

[ASH06] C. Archetti, M. G. Speranza, and A. Hertz, *A Tabu Search Algorithm for the Split Delivery Routing Problem*, Transportation Science **40** (2006).

[BBC+11] M. Benchimol, P. Benchimol, B. Chappert, A. de la Taille, F. Laroche, F. Meunier, and L. Robinet, *Balancing the stations of a self-service bike hiring system*, RAIRO Operations Research **45** (2011), 37–61.

[BGG08] C. Bordenave, M. Gendreau, and Laporte G., *A branch-and-cut algorithm for the preemptive swapping problem*, Networks **59** (2008), 387–399.

[BGL09] C. Bordenave, M. Gendreau, and G. Laporte, *A branch-and-cut algorithm for the non-preemptive swapping problem*, Naval Research Logistics **56** (2009), 478–486.

[CMWC11] D. Chemla, F. Meunier, and R. Wolfler Calvo, *Balancing a bike-sharing system with multiple vehicles*, Odysseus conference, 2011.

[DL91] M. Desrochers and G. Laporte, *Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints*, Operations Research Letters **10** (1991), 27–36.

[EK72] J. Edmonds and R.M. Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Journal of the ACM **19** (1972), 248–264.

[Gal57] T. Gallai, *Maximum-minimum theorems for networks*, Tech. report, 1957.

[GHL94] M. Gendreau, A. Hertz, and G. Laporte, *A tabu search heuristic for the vehicle routing problem*, Management Science **40** (1994), 1276 – 1290.

[GL97] F. Glover and M. Laguna, *Tabu search*, Kluwer Academic, 1997.

[HIMU12] S. Hanafi, A. Ilić, Mladenović, and D. Urošević, *A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem*, European Journal of Operational Research **220** (2012), 270–285.

[HPRMSG09] H. Hernandez Pérez, I. Rodriguez Martin, and J. J. Salazar González, *A hybrid grasp/vnd heuristic for the one-commodity pickup-and-delivery traveling salesman problem*, Computers and Operations Research **36** (2009), 1639–1645.

[HPSG02] H. Hernandez Pérez and J. J. Salazar González, *The one-commodity pickup-and-delivery travelling salesman problem*, Lecture Notes in Computer Science **2570** (2002), 89 – 104.

TABLE 10. Performance of the branch-and-cut cited in Table 1 – **degree branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n20A	20	10	4702	4702.00	4636.50	0.00	0.96	121
n20B	20	10	4769	4769.00	4729.00	0.00	0.58	57
	20	10				0.00	0.15	1691
n20D	20	30	4089	4089.00	4089.00	0.00	0.19	1
n20E	20	30	4299	4299.00	4282.50	0.00	0.2	3
	20	30				0.00	0.16	7
n20B	20	45	3792	3792.00	3792.00	0.00	0.07	1
n20C	20	45	3891	3891.00	3891.00	0.00	0.06	1
	20	45				0.00	0.15	6
n20B	20	1000	3792	3792.00	3792.00	0.00	0.18	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.09	1
	20	1000				0.00	0.18	3
n40B	40	10	5949	5949.00	5686.67	0.00	267.36	19507
n40A	40	10	7169	6418.81	6292.70	11.69	10000.00	466167
	40	10				5.99	9026.36	535641
n40B	40	30	5110	5110.00	5070.67	0.00	3.06	46
n40C	40	30	4644	4644.00	4644.00	0.00	0.95	1
	40	30				0.00	7.98	426
n40B	40	45	5110	5110.00	5040.50	0.00	6.66	337
n40C	40	45	4522	4522.00	4522.00	0.00	0.87	1
	40	45				0.00	2.83	71
n40B	40	1000	5110	5110.00	5040.50	0.00	6.71	77
n40C	40	1000	4522	4522.00	4522.00	0.00	2.24	1
	40	1000				0.00	3.93	21
n60H	60	10	8130	7592.00	7554.00	7.08	10000.00	153855
n60D	60	10	11530	9377.78	9253.01	22.95	10000.00	109576
	60	10				19.67	10000.00	147342
n60J	60	30	6389	6389.00	6212.76	0.00	61.59	1077
n60I	60	30	6390	6209.50	6112.50	2.90	10000.00	272621
	60	30				0.29	1573.97	41027
n60J	60	45	6374	6374.00	6300.45	0.00	57.85	1071
n60A	60	45	5740	5740.00	5685.40	0.00	99.32	1924
	60	45				0.00	89.72	1915
n60D	60	1000	6223	6223.00	6136.67	0.00	169.22	411
n60F	60	1000	5778	5778.00	5726.00	0.00	146.02	653
	60	1000				0.00	247.83	979

TABLE 11. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 10$ and 100 vertices of capacity 20 – **degree branching**

Instance name	**n**	**Q**	**UB2**	**T.t.**	**LB**	**Gap %**
n100F	100	10	14243	11445	10109.80	40.88
n100E	100	10	16261	10847	9965.80	63.17
	100	10		11190		49.20
n100A	100	30	7959	11002	7360.16	8.14
n100J	100	30	9814	11072	7285.04	34.71
	100	30		11092		20.17
n100C	100	45	7998	7849	7950.00	0.60
n100B	100	45	8622	11056	7198.00	19.78
	100	45		10616		7.00
n100E	100	1000	7824	3716	7700.00	1.61
n100B	100	1000	8529	10952	6967.55	22.41
	100	1000		10417		8.09

TABLE 12. Performance of the branch-and-cut cited in Table 3 – **degree branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n100F	100	10	14375	10109.80	10032.50	42.18	10000.03	19571
n100E	100	10	17199	9965.80	9921.62	72.58	10000.00	10211
	100	10				54.01	10000.05	10327
n100A	100	30	8169	7360.16	7223.22	10.98	10000.01	58191
n100J	100	30	9635	7285.04	7195.73	32.25	10000.04	42752
	100	30				20.19	10000.03	50138
n100C	100	45	7950	7950.00	7706.46	0.00	6860.80	33643
n100B	100	45	7886	7198.00	7113.44	9.55	10000.09	60118
	100	45				4.31	9246.13	47180
n100E	100	1000	7700	7700.00	7564.24	0.00	2740.29	339
n100B	100	1000	8235	6967.55	6964.61	18.19	10001.42	1990
	100	1000				8.22	9275.77	1649

TABLE 13. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$, less than 60 vertices and a vertex capacity equal to 60 – **degree branching**

Instance name	**n**	**Q**	**UB2**	**T.t.**	**LB**	**Gap %**
n20B	20	10	9883	58	9883.00	0.00
n20C	20	10	14039	102	14039.00	0.00
	20	10		130		0.00
n20D	20	30	5989	5	5989.00	0.00
n20E	20	30	6245	33	6245.00	0.00
	20	30		10		0.00
n20B	20	45	4174	5	4174.00	0.00
n20C	20	45	5137	7	5113.00	0.47
	20	45		7		0.05
n20B	20	1000	3792	9	3792.00	0.00
n20C	20	1000	4042	9	3891.00	3.88
	20	1000		8		0.84
n40H	40	10	13578	11040	13311.80	2.00
n40F	40	10	15046	11935	14455.10	4.09
	40	10		11856		3.10
n40B	40	30	5949	459	5949.00	0.00
n40F	40	30	7244	10412	6661.40	8.74
	40	30		8732		3.59
n40B	40	45	4879	212	5319.00	0.00
n40C	40	45	5912	901	5912.00	0.00
	40	45		682		0.00
n40B	40	1000	5110	157	5110.00	0.00
n40H	40	1000	4951	236	4772.00	3.75
	40	1000		224		0.94
n60H	60	10	16993	11210	16278.40	4.39
n60C	60	10	23986	11032	17808.80	34.67
	60	10		11053		13.15
n60H	60	30	8123	11010	7782.14	4.38
n60I	60	30	10509	11005	8169.54	28.64
	60	30		11022		14.92
n60H	60	45	6743	3967	6743.00	0.00
n60D	60	45	8842	11167	7689.96	14.98
	60	45		10231		7.36
n60D	60	1000	6223	1050	6223.00	0.00
n60I	60	1000	6355	1469	5914.00	7.45
	60	1000		1130		1.83

[HPSG04a] ———, *A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery*, Discrete Applied Mathematics **145** (2004), 126 – 139.

[HPSG04b] ———, *Heuristics for the one-commodity pickup-and-delivery traveling salesman problem*, Transportation Science **38** (2004), 245–255.

[HPSG07] ———, *The one-commodity pickup-and-delivery travelling salesman problem: inequalities and algorithms*, Networks **4** (2007), 258–272.

[KV02] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 2nd ed., Springer, 2002.

[Min10] A. Mingozzi, personnal communication, 2010.

[RT11] T. Raviv and M. Tzur, personnal communication, 2011.

[RTF09] T. Raviv, M. Tzur, and I.A. Forma, *Static repositioning in a bike-sharing system: Models and solution approaches*, ODYSSEUS, 2009.

[Sch03] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, 2003.

APPENDIX

Proof of Proposition 2. Denote by $\mathcal{Q}$ the set of all target states $\tilde{q}_i$ such that there exists a feasible solution for initial states p_i and target states $\tilde{q}_i$ inducing the sequence. We are interested in the solution of $\min_{\tilde{q}\in\mathcal{Q}} ||\tilde{q} - q||_1$.

(a) We first show that for any solution of $\min_{\tilde{q}\in\mathcal{Q}} ||\tilde{q} - q||_1$, there exist loading and unloading operations compatible with initial and target states p'_i and q'_i, such that

- $0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i
- $\sum_i p'_i = \sum_i q'_i$

and such that $||\tilde{q} - q||_1 = 2\sum_{i\in V}(p_i - p'_i)$.

We have the following central observation: let $\tilde{q}' \in \mathcal{Q}$; there exists $\tilde{q} \in \mathcal{Q}$ such that $||\tilde{q} - q||_1 \leq ||\tilde{q}' - q||_1$ and such that, on each vertex i, there are at least $\max(0, \tilde{q}_i - q_i)$ bikes that have not been moved. Indeed, for each vertex i, choose $\max(0, \tilde{q}'_i - q_i)$ bikes among the $\tilde{q}'_i$ bikes at the end of the route. Consider now the

TABLE 14. Performance of the branch-and-cut cited in Table 5 – **degree branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n20B	20	10	9883	9883.00	9815.00	0.00	0.30	25
n20C	20	10	14039	14039.00	13877.67	0.00	57.43	48393
	20	10				0.00	44.59	32422
n20D	20	30	5989	5989.00	5976.50	0.00	0.41	3
n20E	20	30	6245	6245.00	5928.12	0.00	21.70	14575
	20	30				0.00	3.83	2150
n20B	20	45	4174	4174.00	4174.00	0.00	0.07	1
n20C	20	45	5113	5113.00	4968.00	0.00	1.71	567
	20	45				0.00	1.50	800
n20B	20	1000	3792	3792.00	3792.00	0.00	0.20	1
n20C	20	1000	3891	3891.00	3891.00	0.00	0.09	1
	20	1000				0.00	0.19	2
n40H	40	10	13585	13311.78	13135.01	2.05	9994.00	970628
n40F	40	10	15079	14455.07	14234.90	4.32	9994.00	944740
	40	10				3.25	9995.36	829525
n40B	40	30	5949	5949.00	5789.25	0.00	64.62	9930
n40F	40	30	7244	6661.40	6549.20	8.74	10000.01	840174
	40	30				3.91	8594.44	844082
n40B	40	45	5319	5319.00	5234.25	0.00	7.20	504
n40C	40	45	5912	5912.00	5440.00	0.00	757.16	91073
	40	45				0.00	514.57	82412
n40B	40	1000	5110	5110.00	5076.00	0.00	11.75	393
n40H	40	1000	4772	4772.00	4739.00	0.00	1.17	5
	40	1000				0.00	6.02	142
n60H	60	10	16993	16278.43	16061.96	4.39	10000.00	251254
n60C	60	10	24938	17808.74	17725.78	40.03	10000.00	53851
	60	10				15.49	10000.00	166580
n60H	60	30	8207	7782.14	7674.16	5.46	10000.00	278678
n60I	60	30	10539	8169.53	8098.99	29.00	10000.00	187564
	60	30				17.38	10000.32	232967
n60H	60	45	6743	6743.00	6442.70	0.00	2945.90	125993
n60D	60	45	9061	7689.97	7406.96	17.83	1000.68	241350
	60	45				0.00	9.78	303051
n60D	60	1000	6223	6223.00	6085.39	0.00	100.27	451
n60I	60	1000	5914	5914.00	5729.00	0.00	564.56	5056
	60	1000				0.00	139.00	1020

TABLE 15. Performance of the overall method for $\bar{p}_i = \bar{q}_i = 30$ and 100 vertices of capacity 60 – **degree branching**

Instance name	**n**	**Q**	**UB2**	**T.t.**	**LB**	**Gap %**
n100D	100	10	46600	26720	31803.30	46.52
n100F	100	10	42007	10995	23346.30	79.93
	100	10		15489		65.45
n100F	100	30	14200	11008	10002.20	41.96
n100E	100	30	16491	10880	10049.10	64.10
	100	30		10937		51.54
n100E	100	45	11725	10956	8725.60	34.37
n100B	100	45	15362	10921	9040.11	69.93
	100	45		11035		46.12
n100F	100	1000	7648	3667	7648.00	0.00
n100D	100	1000	8952	11245	7491.00	19.50
	100	1000		7776		5.94

TABLE 16. Performance of the branch-and-cut cited in Table 7 – **degree branching**

Instance name	**n**	**Q**	**UB-BC**	**LB**	**LBr**	**Gap %**	**Time**	**# Nodes**
n100D	100	10	47489	31803.33	31785.56	49.32	10000.03	7262
n100F	100	10	43544	23346.26	23317.48	86.51	10000.00	7121
	100	10				69.01	10000.05	7845
n100F	100	30	14375	10002.20	9928.76	43.71	10000.01	25271
n100E	100	30	16867	10049.18	10027.11	67.85	10000.04	16586
	100	30				56.75	10000.03	15472
n100E	100	45	12028	8725.60	8692.98	37.85	10000.09	61194
n100B	100	45	16153	9040.11	8985.35	78.68	10000.09	35639
	100	45				51.49	10000.29	44260
n100F	100	1000	7648	7648.00	7621.75	0.00	1705.78	2100
n100D	100	1000	7491	7491.00	7364.41	0.00	8924.07	9184
	100	1000				2.04	9275.77	1649

solution $\tilde{q}$ obtained with the same bike displacements except for these bikes, which are not allowed to leave their initial vertices. Denoting by $\bar{p}_i$ the number of bikes on vertex i that are not allowed to move, we have

$$\sum_i \bar{p}_i = \sum_{i:\, q_i < \tilde{q}'_i} \tilde{q}'_i - q_i$$

and

$$\tilde{q}_i = \begin{cases} \tilde{q}'_i + \bar{p}_i & \text{for each vertex } i \text{ such that } q_i \geq \tilde{q}'_i \\ q_i + \bar{p}_i & \text{for each vertex } i \text{ such that } q_i < \tilde{q}'_i \end{cases}$$

Note that $\bar{p}_i \geq \tilde{q}_i - q_i$. The distance to q is equal to

$$||\tilde{q} - q||_1 = \sum_{i:\, q_i \geq \tilde{q}'_i} |\tilde{q}_i - q_i| + \sum_{i:\, q_i < \tilde{q}'_i} |\tilde{q}_i - q_i| \leq \sum_{i:\, q_i \geq \tilde{q}'_i} |\tilde{q}'_i - q_i| + \sum_{i:\, q_i \geq \tilde{q}'_i} \bar{p}_i + \sum_{i:\, q_i < \tilde{q}'_i} \bar{p}_i = \sum_i |\tilde{q}'_i - q_i| = ||\tilde{q}' - q||_1.$$

We can therefore choose the target state $\tilde{q} \in \mathcal{Q}$ minimizing $||\tilde{q} - q||_1$ such that at least $\max(0, \tilde{q}_i - q_i)$ bikes have not been moved for each vertex i. Let us now define, for the route with these target states, p_i^M the number of bikes among the p_i initial ones that have left vertex i and q_i^M the number of bikes that have been brought to vertex i. Note that according to our choice for $\tilde{q}$, we have $q_i^M \leq q_i$ (and of course $p_i^M \leq p_i$). We have

$$||\tilde{q} - q||_1 = \sum_{i \in V} |\tilde{q}_i - q_i| = \sum_{i \in V} |p_i - p_i^M + q_i^M - q_i|. \tag{31}$$

Let $\delta_i := \min(p_i - p_i^M, q_i - q_i^M)$ and define $p'_i := p_i^M + \delta_i$ and $q'_i := q_i^M + \delta_i$ for each vertex i. Note that the initial and target states p'_i and q'_i are feasible in the sense of Proposition 1. According to Equation (31), we have $||\tilde{q} - q||_1 = \sum_{i \in V} |p_i - p'_i + q'_i - q_i|$. Using the fact that for each i, at least one of $p_i - p'_i$ and $q_i - q'_i$ is equal to 0, we get that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

(b) Conversely, assume that we have loading and unloading operations compatible with initial and target states p'_i and q'_i, with

- $0 \leq p'_i \leq p_i$ and $0 \leq q'_i \leq q_i$ for each vertex i
- $\sum_i p'_i = \sum_i q'_i$
- the quantity $\sum_{i \in V} p'_i$ is maximal.

We show that these operations are also compatible with the initial state p and a final state $\tilde{q} \in \mathcal{Q}$ such that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

Since $C = +\infty$, if $p'_i < p_i$ and $q'_i < q_i$, we could have taken into account one more bike on i (increasing by one p'_i), which would have not left vertex i. Therefore, for each i, we have at least one of $p_i - p'_i$ and $q_i - q'_i$ that is equal to 0 and thus, we have $2 \sum_{i \in V} (p_i - p'_i) = \sum_{i \in V} |p_i - p'_i + q'_i - q_i|$. Let us now consider what we have as a final state when we start instead with p_i, while making the same loading and unloading operations: we get $\tilde{q}_i := p_i - p'_i + q'_i$ bikes on each vertex i, for which we have $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$.

According to (b), starting from initial state p, the same loading and unloading operations as for an optimal solution in the sense of Proposition 1 lead to a final state $\tilde{q}$ such that $||\tilde{q} - q||_1 = 2 \sum_{i \in V} (p_i - p'_i)$. (a) shows that this $\tilde{q}$ minimizes $||\tilde{q} - q||_1$ over $\mathcal{Q}$, as required. □ □

Proof of Proposition 6. The proof is very similar to the one of Proposition 5. Again, we work with a reduction from the 2-partition and with the graph of Figure 8. In order to adapt the proof for (RP2), we simulate the $y_{(i,j)}$ by subdividing each arc $a = (i, j)$ in five arcs, introducing four new vertices u_a, u'_a, v_a and v'_a, with $p_{u_a} = 0$, $q_{u_a} = y_{(i,j)}$, $p_{v_a} = m + 1$, $q_{v_a} = 0$, $p_{v'_a} = 0$, $q_{v'_a} = m + 1$, $p_{u'_a} = y_{(i,j)}$, $q_{u'_a} = 0$. Figure 12 illustrates this transformation. The capacity of the vehicle is set to $m+1$. The vertices of this new graph G' provide an instance of the SVOCPDP, and the arcs encode a feasible solution of (RP2) for this instance. The solution satisfies constraints (25) since there are values $y_{(i,j)}$ satisfying constraints (17).

Because of the construction, these values $y_{(i,j)}$ are unique and entirely determined by the initial and target states on each vertex. If there is a feasible solution on the new graph, it will induce a feasible solution on the original graph of Figure 8. This route will imply the existence of a 2-partition, for the same reasons as

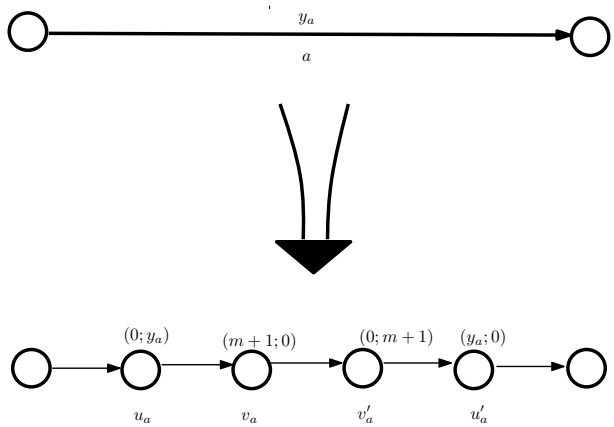

FIGURE 12. Construction of G' in the proofs of Propositions 6, 7, 8

those exposed in the proof of Proposition 5. Conversely, if there is a 2-partition, it implies a feasible solution for the original graph, which will in turn implies a feasible solution for the new graph. □ □

Proof of Proposition 7. We can force the solution built in the proof of Proposition 6 to be optimal, and then the same proof does the job. To force the solution to be optimal, we consider now the graph G' obtained in the proof of Proposition 6 by subdividing each arc in five new arcs. We assume that the cost to traverse each of these arcs is 1. Now, we build from this graph a complete graph for which each arc (i, j) gets for cost the cost of a shortest path between i and j. This complete graph provides an instance of the SVOCPDP. The arc of G' encodes a solution z for (RP2), which is optimal since each new vertex u_a, u'_a, v_a, v'_a has to be visited. □ □

[Proof of Proposition 8. There is only one solution for y once z has been defined according to the proof of Proposition 6, whence the same proof as for Proposition 7 works. □ □

(enpc) CERMICS, UNIVERSITÉ PARIS EST

(Paris13) LIPN (UMR 7030), 99, AV. JEAN-BAPTISTE CLÉMENT, 93430 VILLETANEUSE, FRANCE.