Balancing the stations of a self-service bike hire system

Mike Benchimol, Pascal Benchimol, Benoît Chappert, Arnaud De La Taille, Fabien Laroche, Frédéric Meunier, Ludovic Robinet

RAIRO Operations Research
Will be set by the publisher

BALANCING THE STATIONS OF A SELF SERVICE "BIKE HIRE" SYSTEM

Mike Benchimol[1], Pascal Benchimol[2], Benoît Chappert[3], Arnaud de la Taille[4], Fabien Laroche[5], Frédéric Meunier*[6] and Ludovic Robinet[7]

Abstract. This paper is motivated by operating self service transport systems that flourish nowadays. In cities where such systems have been set up with bikes, trucks travel to maintain a suitable number of bikes per station.

It is natural to study a version of the C-delivery TSP defined by Chalasani and Motwani in which, unlike their definition, C is part of the input: each vertex v of a graph $G = (V, E)$ has a certain amount x_v of a commodity and wishes to have an amount equal to y_v (we assume that $\sum_{v\in V} x_v = \sum_{v\in V} y_v$ and all quantities are assumed to be integers); given a vehicle of capacity C, find a minimal route that *balances* all vertices, that is, that allows to have an amount y_v of the commodity on each vertex v.

This paper presents among other things complexity results, lower bounds, approximation algorithms, and a polynomial algorithm when G is a tree.

Mathematics Subject Classification. ???, ???

1. Self service "bike hire" systems

In 2007, the bike world experienced a great event: the beginning of the Vélib' system in Paris. Vélib' is a self service bike hire system in which bikes are left at the free disposal of users throughout the whole city. Users can take a bike at any time and at any station and replace it at any time and at any station they want, under the conditions that there is bike at the starting station and a free place at the final station.

Such systems already existed before Vélib' but what made Vélib' special is its size, which is largely bigger than previous systems of this kind : 1500 stations, almost 20000 bikes, more than 70000 travels each day. Very quickly, operating such a system has proved difficult. In addition to the servicing, which has to face a quite important number of intentional damages, the stations have to keep a good ratio between the total number of places and the number of bikes in each station (this target ratio is called in Paris "taux de

November 15, 2010.

[1] Université Paris Est, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France. mike.benchimol@eleves.enpc.fr

[2] Ecole Polytechnique, 91128 Palaiseau Cedex, France. pascal.benchimol@polytechnique.edu

[3] Université Paris Est, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France. benoit.chappert@eleves.enpc.fr

[4] Université Paris Est, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France. de-la-ta@eleves.enpc.fr

[5] Ecole Polytechnique, 91128 Palaiseau Cedex, France. fabien.laroche@polytechnique.edu

[6] * *Corresponding author*. Université Paris Est, LVMT, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France. frederic.meunier@enpc.fr

[7] Université Paris Est, ENPC, 6-8 avenue Blaise Pascal, Cité Descartes Champs-sur-Marne, 77455 Marne-la-Vallée cedex 2, France. ludovic.robinet@eleves.enpc.fr

foisonnement", which could be translated by "profusion rate"). Indeed, as it has already been pointed out, if a station becomes empty, potential users won't find bikes; if a station becomes full, a user may to make additional travels in order to find eventually a place to leave the bike. Highly unbalanced situations arise for instance in the morning, when peripherical stations become very quickly empty, and central ones completely saturated.

In the case of Paris, 20 trucks are traveling the city 24 hours a day (except a few hours in the end of the week) in order to maintain the ratio between the total number of places and the number of bikes in each station close to the profusion rate. We call this task, the *balancing* of the stations.

To our knowledge, such systems have not attracted the attention of the researchers yet. However, more and more cities propose such services; other modes, like cars or vans, or other functions, like freight, are envisaged. All these things makes patently obvious that rules or heuristics (at least elementary) for operating such systems should be designed. In this paper, we will deal with the case when no bike are moving. This situation is not completely a simplification. Indeed, every day, at the end of the night, very few bikes are moving. On the other side, it is precisely in the mornings that there are the most intensive use of the system and that the quality of the balancing is important. To start this study, we assume moreover that there is a unique truck.

Finally, note that problems of similar natures can be encountered in other situations. For instance, the caddies of a supermarket must be moved between different points, at the end of the day, in order to get the right number of caddies at each place the customers can take some.

2. Results, notations and plan

2.1. Previous works and results

The problem encountered is a kind of one-commodity pickup-and-delivery problem with one vehicle. A similar problem has been introduced in [8, 9] where there are two kinds of customers: delivery customer, who requires a non-zero amount of the product, and pickup customers, who provides a non-zero amount of the product. Nevertheless, the problem in these papers calls for a minimum distance tour visiting each customer only once. In some case, there is no feasible solution. Hence, it does not suit exactly to the situation described in Section 1.

The swapping problem studied in [2] has not this drawback: a customer can be visited more than once. But in this case, a single vehicle of unit capacity is available for shipping items among the vertices. The swapping problem is to compute a shortest route such that a vehicle can accomplish the rearrangement of the items while following this route. It solves our problem only in a particular case since here the vehicle has unit capacity. It should be emphasized that the swapping problem deals with items of different types, hence it is not a particular version of ours.

Finally, Chalasani and Motwani consider in [4], among other things, the swapping problem for any capacity and only one type of items. They call this problem "the C-delivery TSP". They give an 9.5-approximation algorithm, which we adapt in Section 5 (this ratio has been improved in [1,5]). Nevertheless, their algorithm is polynomial in the capacity C of the truck and in the total number N of items. In our model (Section 3), in our quest of efficiency, we look for a polynomial algorithm in the size of the problem description which is something in $O(\log C + n \log N)$ where n is the number of stations.

The objective of the present paper is to adress the problem described in Section 1, and, as a first approach, to adress it from a theoretical point of view. Among other things, we prove complexity results, describe a 9.5-approximation algorithm working when the truck must come back at its starting point, by an adaptation of the Chalasani-Motwani algorithm (the adaptation consists in replacing a bipartite matching with a bipartite $\boldsymbol{b}$-matching somewhere in the algorithm and in checking that everything remains polynomial), and a 2-approximation algorithm when the network is a complete graph with unit costs and prove that the problem is polynomial when the graph is a tree. We call our problem STATIC STATIONS BALANCING, because of the motivations, which is interesting for its own sake, but it could have been called "1-commodity swapping problem with capacity".

There are two versions of the swapping problem: the *preemptive* one and the *non-preemptive* one. In the preemptive version, the items are *droppable*, i.e. they can be dropped at temporary locations along the route

before being moved to their final destination. In the present paper, we deal with a preemptive version of our problem. Some remarks will be made about the non-preemptive one, mainly in Subsection 8.2, where we prove that the two versions are actually identical in the case of a line.

2.2. Notations and tools

Let F be a finite set and H be a additive semi-group. Let $\boldsymbol{w} \in H^F$. Then for any subset F' of F we denote

$$w(F') = \sum_{f \in F'} w(f).$$

Let $G = (V, E)$ be a graph. For any subset of vertices U, we denote by $\delta(U)$ the set of edges having exactly one endpoint in U. If v is a vertex, we denote $\delta(v)$ instead of $\delta(\{v\})$.

For $\boldsymbol{b} \in \mathbb{Z}_+^V$, a *perfect* $\boldsymbol{b}$*-matching* is a vector $\boldsymbol{m} \in \mathbb{Z}_+^E$ such that for each vertex v of G

$$m(\delta(v)) = b_v. \tag{1}$$

If there is a capacity $\boldsymbol{u} \in \mathbb{Z}_+^E$ on the edges, we can speak of a *capacited* perfect $\boldsymbol{b}$-matching $\boldsymbol{m}$, which requires in addition to Equation (1) that $m_e \leq u_e$ for each edge e.

A necessary condition for the existence of a perfect $\boldsymbol{b}$-matching in a bipartite graph with color classes U and U' is of course that $b(U) = b(U')$.

Optimization problems on $\boldsymbol{b}$-matching in bipartite graphs can be reduced to min-cost flow problems, and hence can be solved in strongly polynomial time.

Let $D = (V, A)$ be a directed graph. For any subset of vertices U, we denote by $\delta^+(U)$ (resp. $\delta^-(U)$) the set of edges leaving (resp. entering) U.

For $\boldsymbol{b} \in \mathbb{Z}_+^V$, a $\boldsymbol{b}$-flow is a vector $\boldsymbol{f} \in \mathbb{R}_+^A$ such that for each vertex v of D

$$f(\delta^+(v)) - f(\delta^-(v)) = b_v.$$

2.3. Plan

In Section 3 we present a model of our problem. Its function is mainly to have a clear definition of our problem. This section is followed by the discussion about the complexity of the problem, which is **NP**-hard in the general case. The following section (Section 5) explains how to adapt the algorithm by Chalasani and Motwani to keep a polynomial time algorithm when the input is not the whole list of bikes and the capacity is not counted in unary basis, but instead when the input is a list of numbers. Section 6 provides an integer linear programming approach, which gives useful lower bounds for our problem. Section 7 deals with the particular case when the graph is a complete graph with unit costs (it can model the case when we only count the loadings and the unloadings of the bikes). In this case, we can solve the problem in polynomial time for fixed capacity $C = 1$ or $C = 2$ of the vehicle. There is also an easy (greedy) 2-approximation algorithm in this case. Finally, we describe a polynomial time algorithm solving the problem exactly when the graph is a tree (Section 8). The last section (Section 9) is devoted to several open questions.

3. The model

We are given a graph $G = (V, E)$ and a *capacity* $C \geq 0$. A *state* s is a couple $(\boldsymbol{x}, p)$ where $\boldsymbol{x} \in \mathbb{R}_+^V$ and p a vertex in V. Two states $s = (\boldsymbol{x}, p)$ and $s' = (\boldsymbol{y}, q)$ are said to be *adjacent* if we have simultaneously

- $x_v = y_v$ for all $v \notin \{p, q\}$,
- $pq \in E$,
- $x_p - y_p = y_q - x_q$ and
- $|x_p - y_p| \leq C$.

A *move* consists in going from a state s to an adjacent state s'. If the graph is endowed with a *cost* $\boldsymbol{c} \in \mathbb{R}_+^E$, the *cost* of a move between two adjacent states $s = (\boldsymbol{x}, p)$ and $s' = (\boldsymbol{y}, q)$ is simply $c(pq)$. The cost of a sequence of moves is the sum of the cost of the moves of the sequence.

The interpretation of these definitions is obvious. A state $s = (\boldsymbol{x}, p)$ encodes the position p of the truck and the number of bikes x_v on each station v. The number x_p can be alternatively seen as the number of bikes on p plus the number of bikes in the truck or as the total number of bikes on p with the additional rule that when the truck arrives on a station, it unloads all its bikes. We are in the preemptive version, whence it doesn't matter.

A move corresponds to a real move of the truck. If the two adjacent states are $s = (\boldsymbol{x}, p)$ and $s' = (\boldsymbol{y}, q)$, the number of bikes in the truck during the move is $x_p - y_p$.

The problem we are interested in consists in going from an initial state $i = (\boldsymbol{x}, p)$ to a terminal state $t = (\boldsymbol{y}, q)$, using only moves between adjacent states, with a sequence of moves of minimal cost. A vertex (or a station) is said to be *in excess* (resp. *in default*) if $x_v > y_v$ (resp. $x_v < y_v$). A vertex or a station such that $x_v = y_v$ is *balanced*. The set of balanced stations is denoted by $B(\boldsymbol{x}, \boldsymbol{y})$.

The same notions hold for a subset of vertices: a subset $U \subseteq V$ is said to be in excess, or in default, or balanced, depending on whether $x(U) - y(U)$ is > 0, < 0 or $= 0$.

We define U^{ex} to be the set of vertices such that $x_v > y_v$ and U^{def} to be the set of vertices such that $x_v < y_v$.

Assumption.

We assume in the sequel that $x(V) = y(V)$, otherwise there is no solution.

STATIC STATIONS BALANCING PROBLEM

Instance : A graph $G = (V, E)$, a cost function $\boldsymbol{c} \in \mathbb{R}_+^E$, a capacity C and two states $i = (\boldsymbol{x}, p)$ and $t = (\boldsymbol{y}, q)$.

Task : Find the minimal cost of a sequence of moves that allows to go from the state i to the state t.

This formulation is apparently innocent, but even before discussing whether it is possible to solve it in polynomial time or not, a remark must be done.

The minimum cost l, solution of the above problem, has necessarily an encoding that is polynomial in the size of the input: indeed, for obvious reasons, we have $l \leq nx(V)c(E)$. But it is easy to find instances for which the encoding of the sequence realizing the optimum is not polynomial in the size of the input: take the graph G with two vertices u, v and a unique edge connecting these two vertices and define $C = 1$. Consider the initial state $i = (\boldsymbol{x}, p)$ and the terminal state $t = (\boldsymbol{y}, q)$ with $x_u = 2\alpha$, $x_v = 0$, $y_u = \alpha$ and $y_v = \alpha$, for some positive real number α. The minimum number of moves is $2\alpha - 1$. The size of the input is $O(\log \alpha)$ and the size of the sequence of moves realizing the optimum is in $O(\alpha)$, which is exponential in the size of the input. Whether there is always a way to have an encoding of an optimal sequence whose size is polynomial in the size of the input is an open question. See the discussion of Section 9.

If we want to have an explicit solution, a way to circumvent these kinds of problem is simply to ask for an output that, in addition to the length of the optimal sequence, gives its first move. Indeed, the STATIC STATIONS BALANCING PROBLEM (SSBP) has this nice property to remain a STATIC STATIONS BALANCING PROBLEM after a move. Hence, we can restate it under the following form.

STATIC STATIONS BALANCING PROBLEM – research version

Instance : A graph $G = (V, E)$, a cost function $c \in \mathbb{R}_+^E$, a capacity C and two states $i = (\boldsymbol{x}, p)$ and $t = (\boldsymbol{y}, q)$.

Task : Find the minimal cost of a sequence of moves that allows to go from the state i to the state t and the first move of such an optimal sequence.

4. Complexity results

The main message of this section is that the SSBP is **NP**-hard, even in restricted cases. Indeed, the PARTITION PROBLEM and the HAMILTONIAN CYCLE IN BIPARTITE GRAPH PROBLEM can be straightforwardly reduced to it. We give the full details for sake of completness.

Theorem 1. *The SSBP is* **NP***-hard even for a complete graph with unit costs and a* $\boldsymbol{y}$ *constant on* V*.*

Proof. Indeed, the PARTITION PROBLEM can be reduced in polynomial time to the SSBP for a complete graph with unit costs and a $\boldsymbol{y}$ constant on V. Let $a_1, \ldots, a_n$ be an instance of the PARTITION PROBLEM. This latter problem is an **NP**-complete decision problem which asks whether it is possible to find $A \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in A} a_j = \sum_{j \notin A} a_j$ ([7] for further comments).

We describe now the reduction. Take the complete graph $K_{n+1} = (V, E)$ with $n+1$ vertices. The vertices are identified with the integers $1, 2, \ldots, n+1$. Define $r := 1 + \max_{j \in \{1,\ldots,n\}} a_j$. Consider the SSBP for the complete graph with unit costs, a capacity $C = \frac{1}{2} \sum_{j=1}^{n} a_j$ (w.l.o.g. we can assume that $\sum_{j=1}^{n} a_j$ is even), and an initial and a terminal states such that $x_j = r - a_j$ for $j \in \{1, \ldots, n\}$, $x_{n+1} = r + \sum_{j=1}^{n} a_j$, $p = q = n+1$, and $y_j = r$ for all $j \in \{1, \ldots, n+1\}$.

If the answer to the PARTITION PROBLEM is yes, the optimal solution to the SSBP requires exactly $n+2$ moves. Indeed, in this case, there is a subset A such that $\sum_{j \in A} a_j = C$; the truck starts from vertex $n+1$, visits first the vertices $j \in A$, comes back to the vertex $n+1$, finishes its tour by visiting the vertices $\notin A$ and then goes back to vertex $n+1$.

If the optimal solution to the SSBP requires exactly $n+2$ moves, the answer to the PARTITION PROBLEM is yes. Indeed, each vertex $\in \{1, \ldots, n\}$ must be visited at least once, and the truck must leave the vertex at least twice (since there are $2C$ bikes in excess). If it requires exactly $n+2$ moves, then each vertex $\in \{1, \ldots, n\}$ is visited exactly once, and the vertex $n+1$ is left exactly twice, whence inducing a partition of the vertices in $\{1, \ldots, n\}$ in two parts A and B (the vertices visited before the second leaving of vertex $n+1$ and the vertices visted after the second leaving of vertex $n+1$). We have then necessarily $\sum_{j \in A} a_j \leq C$ and $\sum_{j \in B} a_j \leq C$ since the vertices are balanced with only one visit of the truck. Since $\sum_{j \in A \cup B} a_j = 2C$, we have $\sum_{j \in A} a_j = \sum_{j \in B} a_j$ with A and B providing a partition of $\{1, \ldots, n\}$. □

Theorem 2. *The SSBP is* **NP***-hard even for a bipartite graph with unit costs, fixed* C *(even* $C = 1$*) and* $y_v = 1$ *for all* $v \in V$*.*

Proof. Indeed, the HAMILTONIAN CYCLE IN BIPARTITE GRAPH PROBLEM can be reduced in polynomial time to the SSBP in a bipartite graph, with unit costs, C fixed and $y(v) = 1$ for all $v \in V$. Let $G = (V, E)$ be a bipartite graph. The HAMILTONIAN CYCLE IN BIPARTITE GRAPH PROBLEM is an **NP**-complete decision problem asking whether there is an hamiltonian cycle in a bipartite graph ([7] for further comments).

We describe now the reduction. We suppose $C \geq 1$ fixed. Set $y_v = 1$ for all $v \in V$. Let A, B be the two color classes of G. Define $x_v = 2$ for all $v \in A$ and $x_v = 0$ for all $v \in B$. Define moreover $p = q$ to be any vertex of A. Now if there is an hamiltonian path in G, then the optimal solution to the SSBP is n moves, where n is the number of vertices (each move from A to B consists in removing a bike in excess in a station of A and in putting it on a station of B). Otherwise, the optimal solution needs at least $n+1$ moves since each station must be visited at least once (no station is balanced in the initial state) and there is no hamiltonian cycle. □

5. A 9.5-approximation algorithm

In this section, we prove that it is possible to adapt the algorithm by Chalasani and Motwani [4] in order to get a 9.5-approximation algorithm that works in polynomial time for the SSBP when $p = q$ (that is when the truck starts and finishes at the same vertex). Recall that their algorithm was designed for the C-DELIVERY TRAVELING SALESPERSON PROBLEM. The SSBP is the C-DELIVERY TRAVELING SALESPERSON PROBLEM when the number C is a part of the input. Their algorithm is polynomial in C, but not in $\log C$, as it should if we want to have a polynomial algorithm for SSBP.

Fortunately, the algorithm can be easily adapted (mainly, a bipartite matching is replaced by a bipartite $\boldsymbol{b}$-matching). We get not only a 9.5-approximate value of an optimal balancing tour, but also a polynomial description of a tour that realizes this value (the tour itself has not necessarily a polynomial size since some edges could be traversed a number of times of the same order than C).

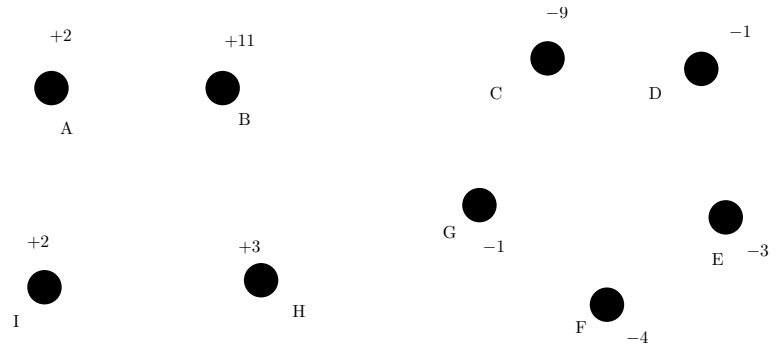

FIGURE 1. An example of input. The number next each vertex is b_v if v is in excess, and $-b_v$ if v is in default.

To make the adaptation working, we need the following lemma. Recall that an Eulerian graph is a connected graph whose vertices have all an even degree, and an Eulerian tour is a closed walk traversing each edge exactly once. An Eulerian graph has always an Eulerian tour.

Lemma 1. *Let H be an Eulerian bipartite graph, with color classes U and U'. Consider an input of the SSBP on this graph, with*

- *a capacity C of the truck,*
- *$q = p \in U$,*
- *$x_v = \frac{1}{2}C\deg(v)$ if $v \in U$ and $x_v = 0$ if not.*
- *$y_v = 0$ if $v \in U'$ and $y_v = \frac{1}{2}C\deg(v)$ if not.*

Then H can be balanced by any Eulerian tour starting and finishing at $p = q$.

Proof. Take any Eulerian tour starting and finishing at p. Apply the following rule: each time the truck enters a vertex of U, it takes exactly C bikes; each time the truck enters a vertex of U', it delivers exactly C bikes on this vertex. Assuming that there is no problem with the capacity of the truck, this algorithm balances all the vertices, since the truck takes exactly $\frac{1}{2}\deg(v)$ times C bikes from each vertex of U, and puts exactly $\frac{1}{2}\deg(v)$ times C bikes on each vertex of U'. It remains to check that there is no problem with the capacity of the truck.

We check by induction on the number of traversed edges that each time the truck leaves a vertex of U, it has exactly C bikes, and each time it leaves a vertex of U', it has no bike. This property is true for the first edge of the Eulerian tour. Now, assume it is true at a moment on the tour.

If the truck leaves a vertex of U, it has by induction exactly C bikes. Since H is bipartite, it arrives now on a vertex of U', it delivers exactly C bikes and has now no bike.

If the truck leaves a vertex of U', it has by induction no bike. Since H is bipartite, it arrives now on a vertex of U, it takes exactly C bikes and has now exactly C bikes. □

Let us now describe how to adapt the algorithm by Chalasani and Motwani. Recall that we need to suppose that $p = q$.

We assume that G is the complete graph with a cost function satisfying the triangle inequality. We assume moreover without loss of generality that $x(U^{ex}) - y(U^{ex})$ is a multiple of C (it can be achieved with a dummy vertex). Finally, we assume that $x_p = y_p = 0$, which is not a restriction, since we can add a dummy vertex at distance 0 of the original p vertex.

Let $\boldsymbol{m}$ be a minimum-cost perfect $\boldsymbol{b}$-matching of the bipartite graph induced by G when we keep only edges connecting a vertex in default to a vertex in excess, where $b_v := |x_v - y_v|$. On Figure 1, we see an input of the problem, and on Figure 2 an illustration of $\boldsymbol{m}$.

We first use Christofides' heuristic [6] to obtain a 1.5-approximate tour T^{ex} on the set of the vertices in excess. Next, we decompose T^{ex} into consecutive paths $P_1^{ex}, P_2^{ex}, \ldots$ such that either

- $x(P_i^{ex}) - y(P_i^{ex}) = C$ or

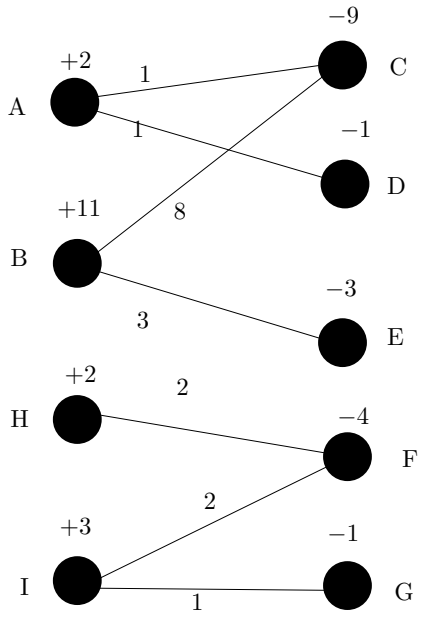

FIGURE 2. A perfect $\boldsymbol{b}$-matching for the input of Figure 1.

- $x(P_i^{ex}) - y(P_i^{ex}) = a_i^{ex} C$ for some integer a_i^{ex} if P_i^{ex} is a single vertex,

by deleting a set of edges that are spaced along the tour. We proceed as follows.

Start at an arbitrary vertex of T^{ex} and follow the tour in one direction until the sum of the b_v of the vertices encountered is equal to or larger than C. These vertices provide then P_1^{ex}. If the sum of the b_v is not a multiple of C, we make a copy of the last vertex v of P_1^{ex} and insert it just after v in the tour (the new edge has cost 0). This new vertex receives just enough bikes from v in order to get a multiple of C bikes on P_1^{ex}. We transform accordingly $\boldsymbol{m}$ in order to still have a perfect $\boldsymbol{b}$-matching (the new vertex 'takes' from each m_e with $e \in \delta(v)$ an arbitrary part, while having the right value for $m(\delta(v))$).

Then we do again the same procedure, starting from the next vertex of the tour (if a copy of v has been done, this copy is this next vertex). And so on, until the whole tour T^{ex} has been decomposed.

We define similarly a 1.5-approximate tour T^{def} on the set of vertices in default, the paths $P_1^{def}, P_2^{def}, \ldots$ and the integers a_j^{def}.

Because of the possible splitting of vertices, they may be more than n vertices at the end, but not more than $3n$ (the 3 comes from the fact that a vertex can be split in 3 copies when it has strictly more than C bikes and when it arrives at the end of a path P_i^{ex}).

The whole procedure is illustrated on Figures 3 and 4.

The paths P_i^{ex} and P_j^{def} are seen as "super-node". We now overlay $\boldsymbol{m}$. We get a bipartite graph with integers m_e on each edge e such that $m(\delta(P_i^{ex})) = C$ for all i, $m(\delta(P_j^{def})) = C$ for all j, except when P_i^{ex} or P_j^{def} is a single vertex in which case $m(\delta(P_i^{ex}))$ or $m(\delta(P_j^{def}))$ can be any multiple of C. We pick a minimal cost perfect $\boldsymbol{b'}$-matching $\boldsymbol{m'}$ of this bipartite graph, where $b'_{P_i^{ex}} := 1$ if P_i^{ex} has at least two vertices, or $b'_{P_i^{ex}} := a_i^{ex}$ if P_i^{ex} is a single vertex. The same holds for the P_j^{def}. Clearly $c(\boldsymbol{m'}) \leq \frac{1}{C} c(\boldsymbol{m})$ (we use the fact that the edges of a C-regular bipartite graph can be partitioned into C perfect matching – it is König's theorem [11]). On Figure 5, we see an example of a $\boldsymbol{b'}$-matching $\boldsymbol{m'}$ and how we can build a solution from it on Figure 6.

At this stage we have a collection of connected subgraphs $H_1, H_2, \ldots$ each consisting of one or several super-nodes in excess connected to one or several super-nodes in default. According to Lemma 1, each of this

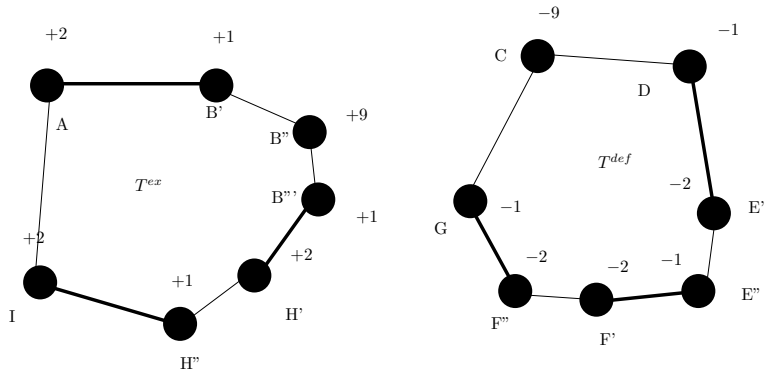

FIGURE 3. Construction of the paths in the 9.5-approximation algorithm for $C = 3$.

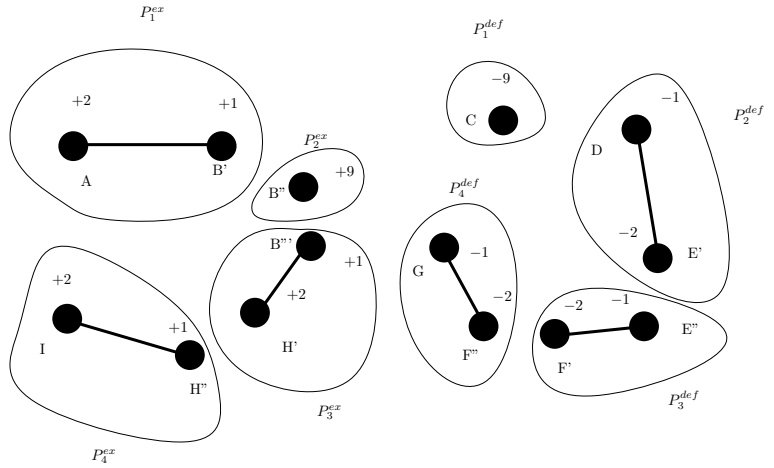

FIGURE 4. Construction of the super-nodes in the 9.5-approximation algorithm for $C = 3$.

subgraph can be balanced with its own bikes, starting and finishing on one vertex of one of its paths in excess, using $2m'_e$ times each edge e in $\boldsymbol{m}'$, and twice the edges of its paths. The lemma can be applied since making $2m'_e$ copies of each edge e in any H_i makes it an Eulerian graph. The sum of the costs for the balancing of all vertices in the H_i is then $\leq 2c(T^{ex}) + 2c(T^{def}) + 2c(\boldsymbol{m}')$.

To go from one of these subgraphs H_i to another one, we can use edges of T^{ex}, which give an additional cost $\leq c(T^{ex})$.

We can balance the whole graph G with a tour of cost SOL such that

$$\begin{array}{rcl} \text{SOL} & \leq & 2c(T^{ex}) + 2c(T^{def}) + 2c(\boldsymbol{m}') + c(T^{ex}) \\ & \leq & 3c(T^{ex}) + 2c(T^{def}) + \frac{2}{C}c(\boldsymbol{m}) \\ & \leq & 7.5\text{OPT} + 2\text{OPT} \\ & \leq & 9.5\text{OPT} \end{array}$$

Here we use that $\frac{1}{C}c(\boldsymbol{m}) \leq \text{OPT}$, which is proved in [4].

According to the discussion we have just made

Theorem 3. *There is a polynomial time algorithm that gives a* 9.5*-approximate solution for the SSBP when* $p = q$ *(the truck starts and finishes at the same vertex).*

Proof. The only thing that has not be treated in the discussion above concerns the fact that the truck starts and finishes in p. Actually, p (which is assumed w.l.o.g. to be such that $x_p = y_p = 0$) can be considered

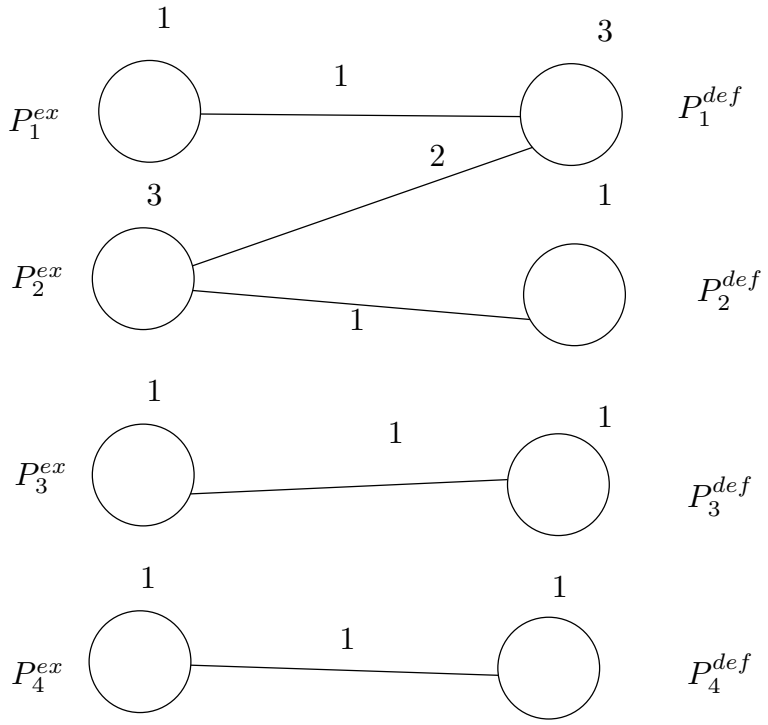

FIGURE 5. A $\boldsymbol{b}'$-matching $\boldsymbol{m}'$ as used in the 9.5-approximation algorithm (the numbers next the super-nodes are the b'_v).

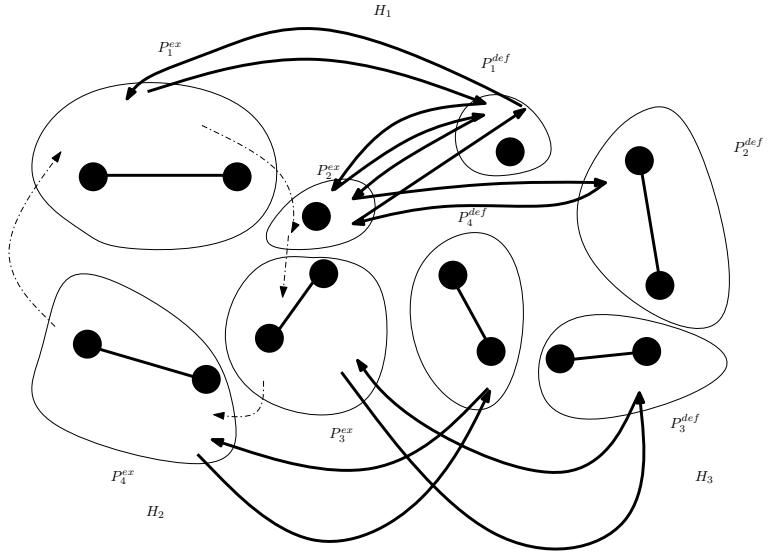

FIGURE 6. A solution for the SSBP induced by the $\boldsymbol{b}'$-matching $\boldsymbol{m}'$ of Figure 5.

arbitrarily as a vertex in excess. Since, in the proof, we can choose the vertex in excess which we start with (in the definition in P_1^{ex}), the above algorithm is definitely a 9.5-approximation algorithm. □

Remark. If $p \neq q$, we can get with the same method another approximation, by replacing the tours T^{ex} and T^{def} by p-q paths on the set $U^{ex} \cup \{p, q\}$ and $U^{def} \cup \{p, q\}$. But this time, the 1.5 ratio of Christofides heuristic does not work and we only get a 5/3-approximation (see [10]), leading finally to a 31/3-approximation. We do not know if it is possible to do better.

6. Lower bounds

In this section, we provide a lower bound for the SSBP. It is the optimal value of an integer linear program, whose variables "count" the number of times the truck passes on each edge. The constraints are of three kinds:

- the Eulerian condition: except maybe in p and q, the truck enters and leaves each vertex an even number of times. They are labelled (ii).
- "subtour elimination", like for the TSP: if the truck is in $p \in U \subseteq V$ and they are unbalanced stations in $\bar{U}$, then it must necessarily traverse $\delta(U)$ at least once, and maybe more depending on the position of q and the other unbalanced stations. They are labelled (iii). A useful notation to write down these constraints is

$$\mu(p,q,U,\boldsymbol{x},\boldsymbol{y}) := \begin{cases} 0 & \text{if } p,q \in U \text{ and } \bar{U} \subseteq B(\boldsymbol{x},\boldsymbol{y}) \\ 0 & \text{if } p,q \in \bar{U} \text{ and } U \subseteq B(\boldsymbol{x},\boldsymbol{y}) \\ 1 & \text{if } p \in U \text{ and } q \in \bar{U} \\ 1 & \text{if } p \in \bar{U} \text{ and } q \in U \\ 2 & \text{if } p,q \in U \text{ and } \bar{U} \setminus B(\boldsymbol{x},\boldsymbol{y}) \neq \emptyset \\ 2 & \text{if } p,q \in \bar{U} \text{ and } U \setminus B(\boldsymbol{x},\boldsymbol{y}) \neq \emptyset. \end{cases}$$

- "capacity constraints", like for the CAPACITED VEHICLE ROUTING problem: if $U \subseteq V$ has too many bikes, the truck must leaves U enough times to remove all these bikes in excess. They are labelled (iv). A useful notation to write down these constraints is

$$\eta(p,q,U) := \begin{cases} -1 & \text{if } p \in U \text{ and } q \in \bar{U} \\ 0 & \text{if } p \in U \text{ and } q \in U \\ 0 & \text{if } p \in \bar{U} \text{ and } q \in \bar{U} \\ +1 & \text{if } p \in \bar{U} \text{ and } q \in U \end{cases}$$

The optimal value of the following integer linear program

$$\begin{array}{lllll} \text{Min}_{\boldsymbol{z}} & \sum_{e\in E} c_e z_e & & & \\ \text{s.t.} & z_e \in \mathbb{Z}_+ & \text{for all } e \in E & \text{(i)} & \\ & z(\delta(v)) \text{ is even} & \text{for all } v \in V \setminus \{p,q\} & \text{(ii)} & \\ & z(\delta(U)) \geq \mu(p,q,U,\boldsymbol{x},\boldsymbol{y}) & \text{for all } U \subseteq V,\, U \neq \emptyset & \text{(iii)} & \\ & z(\delta(U)) \geq 2\left\lceil \frac{x(U)-y(U)}{C} \right\rceil + \eta(p,q,U) & \text{for all } U \subseteq V,\, U \neq \emptyset & \text{(iv)} & \end{array} \tag{2}$$

is clearly a lower bound for the SSBP.

We will see that it is not at all easy to solve this program. Lower bounds that are easier to compute can be obtained by deleting some of the constraints. Of course, the bounds can then be worse. In Sections 7 and 8, the quality of the solutions will be proved with bounds obtained by deleting constraint (ii). Program (2) is an integer linear program: the constraint (ii) can be rewritten in $z(\delta(v)) = 2y_v$ for a new integer variable $\boldsymbol{y} \in \mathbb{Z}_+^V$.

A standard way to solve this kind of program is the branch-and-bound technic, or more precisely here, since the number of constraints is exponential, the branch-and-cut technic. Hence, we must check whether it is possible to separate easily constraints (iii) and (iv). The separation problem for (iii) is: given a map $z : E \to \mathbb{R}_+$, can we find a subset $U \subseteq V$ with $U \setminus B \neq$ and $\bar{U} \setminus B \neq \emptyset$ such that $z(\delta(U)) < 1$? This is simply a min-cut problem, thus there is no problem.

Unfortunately, for the constraint (iv), it is not clear whether there is a polynomial time routine that can separate it. In the classical formulation of the CAPACITED VEHICLE ROUTING PROBLEM – or for short CVRP – there is a similar constraint and its complexity status is open (see [3] ; the same paper proposes an efficient heuristic to separate this constraint). The similar constraint of the CVRP is

$$x(\delta(S)) \geq 2\left\lceil \frac{d(S)}{C} \right\rceil \text{ for all } S \subseteq V, S \neq \emptyset,$$

where $\boldsymbol{x} \in \mathbb{R}_+^E$ is the variable, $\boldsymbol{d} \in \mathbb{R}_+^V$ is the *demand function* and C the capacity of the truck.

Instead of separating

$$z(\delta(U)) \geq 2 \left\lceil \frac{x(U) - y(U)}{C} \right\rceil + \eta(p, q, U) \quad \text{for all } U \subseteq V,$$

we can separate a weaker constraint (and thus have a weaker bound) in polynomial time

$$z(\delta(U)) \geq 2\frac{x(U) - y(U)}{C} + \eta(p, q, U) \quad \text{for all } U \subseteq V.$$

This latter constraint can be separated in polynomial time, for it can be reformulate as a min-cut problem as well.

If an optimal solution for the program (2) can be transformed in an optimal solution of the SSBP, then the number of bikes transferred by each move is a $\boldsymbol{b}$-flow of the digraph obtained from G while replacing each edge e by z_e arcs oriented according a p-q Eulerian walk, with a capacity C on each arc, and $b_v = x_v - y_v$ for all vertices $v \in V$. Such a flow always exists since constraint (iv) implies that the cut condition is satisfied everywhere in the digraph. But it is not clear, even if an optimal solution for the program (2) can be transformed in optimal solution of the SSBP, that any $\boldsymbol{b}$-flow can be such a solution.

At least, solutions of program (2) can be used in heuristics to build reasonably good solutions.

7. Complete graph with unit costs

The case of a complete graph with unit cost can model the case when we only want to count the number of times we load or unload bikes on a station.

7.1. A 2-approximation algorithm

We show in this subsection that a greedy algorithm provides to the SSBP for unit costs on the complete graph a 2-approximation algorithm.

Consider the following algorithm.

Repeat these two tasks until all stations are balanced:

- While the truck is not full and it remains at least one vertex in excess, repeat: go to a station v in excess, take as many bikes as possible without exceeding the capacity of the truck and while leaving at least y_v bikes on the station.
- While the truck is not empty, repeat: go to a station v in default and put as many bikes as possible while leaving at most y_v bikes on the station.

When they are all balanced, add a move to reach the terminal station.

Theorem 4. *The greedy algorithm described above provides a solution* SOL *such that* $\text{SOL} \leq 2\text{OPT} + 1$*, where* OPT *denotes the optimal value of the SSBP.*

Proof. Inside the proof, we call the quantity $\sum_{v \in V} |x_v - y_v|$ the *total imbalance*.

We assume first that we start on a station in excess.

We prove by induction on the total imbalance that

$$\text{SOL} \leq 2 \left(\sum_{v \in V} \left\lceil \frac{|x_v - y_v|}{C} \right\rceil \right) - 2. \tag{3}$$

If $\sum_{v \in V} |x_v - y_v| = 2$, the inequality (3) is satisfied.

Now, suppose that $\sum_{v \in V} |x_v - y_v| \geq 3$. Denote by $s_1, \ldots, s_g$ the first g stations in excess followed by the first h stations $t_1, \ldots, t_h$ in default when we apply the greedy algorithm above. Let $\alpha_i := x_{s_i} - y_{s_i}$ and

$\beta_j := y_{t_j} - x_{t_j}$. Note that if $g \geq 2$, we have

$$\sum_{i=1}^{g-1} \alpha_i < C, \tag{4}$$

and a similar property holds for h and the β_j.

The truck strats at a vertex is excess, visits $g-1$ more stations in excess, h stations in default, goes to a station in excess and then is in a situation similar to the one it faced at the beginning, but with a stricly smaller total imbalance (hence induction can be used). Each station s_i for $i \leq g-1$ has now the right number of bikes. The station s_g has now $x_{s_g} - (C - \sum_{i=1}^{g-1} \alpha_i)$ bikes. Each station t_j for $j \leq h-1$ has now the right number of bikes. The station t_h has now $x_{t_h} + (C - \sum_{j=1}^{h-1} \beta_j)$ bikes.

Using induction, after the visit of the g stations in excess and the h stations in default, the trucks balances the remaining with a cost smaller than

$$2\left(\sum_{v\in V\setminus\{s_1,\dots,s_g,t_1,\dots,t_h\}} \left\lceil \frac{|x_v - y_v|}{C} \right\rceil\right) + 2\left\lceil \frac{|x_{s_g} - (C - \sum_{i=1}^{g-1}\alpha_i) - y_{s_g}|}{C} \right\rceil + 2\left\lceil \frac{|x_{t_h} + C - \sum_{j=1}^{h-1}\beta_j - y_{t_h}|}{C} \right\rceil - 2.$$

Thus

$$\mathrm{SOL} \leq g + h + 2\left(\sum_{v\in V\setminus\{s_1,\dots,s_g,t_1,\dots,t_h\}} \left\lceil \frac{|x_v - y_v|}{C} \right\rceil\right) + 2\left\lceil \frac{\sum_{i=1}^{g}\alpha_i - C}{C} \right\rceil + 2\left\lceil \frac{\sum_{j=1}^{h}\beta_j - C}{C} \right\rceil - 2.$$

We have

$$g + 2\left\lceil \frac{\sum_{i=1}^{g}\alpha_i}{C} \right\rceil \leq 2g + 2\left\lceil \frac{\alpha_g}{C} \right\rceil.$$

Indeed, for $g = 1$, it is obvious, and for $g \geq 2$, it is a consequence of (4). A similar inequality is satisfied for h and the β_j.

Hence,

$$\mathrm{SOL} \leq 2\left(\sum_{v\in V\setminus\{s_1,\dots,s_g,t_1,\dots,t_h\}} \left\lceil \frac{|x_v - y_v|}{C} \right\rceil\right) + 2\left\lceil \frac{\alpha_g}{C} \right\rceil + 2\left\lceil \frac{\beta_h}{C} \right\rceil + 2(g-1) + 2(h-1) - 2,$$

which proves equation (3).

To conclude, note that according to program (2)

$$\mathrm{OPT} \geq \sum_{v\in V} \left\lceil \frac{|x_v - y_v|}{C} \right\rceil \quad \text{(if } q \text{ is in default, add a } -1 \text{ on the right-hand side)},$$

which prove that $\mathrm{SOL} \leq 2\mathrm{OPT}$.

Add a move to deal with the general case. □

Although SOL is computable in linear time once we have fixed any order on the vertices, the algorithm itself is not polynomial, for the reasons explained in Section 4.

7.2. Polynomial algorithms for fixed C

The SSBP is polynomial for a complete graph with unit costs and fixed $C = 1, 2$. Note if we remove the condition of unit costs the problem is **NP**-hard again since the problem on a bipartite graph, which is **NP**-hard according to Theorem 2, can be translated into a problem on a complete graph where the costs on an edge uv is the minimal number of edges of a path between u and v in the bipartite graph.

Case $C = 1$.

Repeat this two tasks alternately until all stations are balanced:

- Choose a station v in excess and take 1 bike from it.
- Choose a station v in default and put 1 bike on it.

Case $C = 2$.

Repeat this two tasks alternately until all stations are balanced:

- Choose a station v in excess with $x_v - y_v \geq 2$ and take 2 bikes; if there is no such station choose two stations v and w in excess with $x_v - y_v = 1$ and $x_w - y_w = 1$ and take these two bikes; if there is no such stations, take the last bike in excess if there is one.
- Choose a station v in default with $y_v - x_v \geq 2$ and put 2 bikes on it; if there is no such station choose two stations v and w in default with $y_v - x_v = 1$ and $y_w - x_w = 1$ and fill in these two bikes; if there is no such stations, fill in the last bike in default if there is one.

Proposition 1. *Assume without loss of generality that p and q are unbalanced (otherwise, it is easy to add the right number of moves to an optimal solution with p or q balanced). The (non-polynomial but greedy) algorithms described above solve the SSBP for fixed $C \in \{1, 2\}$ on the complete graph with unit costs. The number of moves required by them are*

- *if $C = 1$:*
$$\sum_{v \in V} |x_v - y_v| + \eta(p, q, U^{ex}).$$
- *if $C = 2$:*
$$\sum_{v \in V} \left\lceil \frac{|x_v - y_v|}{2} \right\rceil + \eta(p, q, U^{ex}).$$

Note that the SSBP in its research version can be solved in polynomial time, for we just have to compute one of the formulas and to give the first step of one of the algorithms described above.

Proof. The case $C = 1$ is obvious.

Proof of the case $C = 2$. Define respectively U_0^{ex} and U_1^{ex} to be the set of vertices $v \in U^{ex}$ such that $x_v - y_v = 0 \mod 2$ and $x_v - y_v = 1 \mod 2$. Define also U_0^{def} and U_1^{def} to be the set of vertices $v \in U^{def}$ such that $x_v - y_v = 0 \mod 2$ and $x_v - y_v = 1 \mod 2$.

The number of moves generated by the algorithm is

$$\sum_{v \in U_0^{ex}} \frac{x_v - y_v}{2} + \sum_{v \in U_1^{ex}} \left\lceil \frac{x_v - y_v}{2} \right\rceil + \sum_{v \in U_0^{def}} \frac{y_v - x_v}{2} + \sum_{v \in U_1^{def}} \left\lceil \frac{y_v - x_v}{2} \right\rceil + \eta(p, q, U^{ex}),$$

which can be rewritten more compactly

$$\sum_{v \in V} \left\lceil \frac{|x_v - y_v|}{2} \right\rceil + \eta(p, q, U^{ex}).$$

The algorithm balances all the stations: indeed, except maybe for the last move or the last two moves, when the truck enters U^{def}, it carries 2 bikes, and when it leaves U^{def}, it is empty.

The following program is program (2) with constraints missing (the last one is obtained formally by letting $U := V \setminus \delta(v)$ in (iv) of (2)),

$$\begin{array}{lll} \mathrm{Min}_{\boldsymbol{z}} & \sum_{e \in E} z_e & \\ \text{s.t.} & z_e \in \mathbb{Z}_+ & \text{for all } e \in E \\ & z(\delta(v)) \geq 2 \left\lceil \frac{x_v - y_v}{2} \right\rceil + \eta(p, q, v) & \text{for all } v \in U^{ex} \\ & z(\delta(v)) \geq 2 \left\lceil \frac{y_v - x_v}{2} \right\rceil - \eta(p, q, v) & \text{for all } v \in U^{def} \end{array} \tag{5}$$

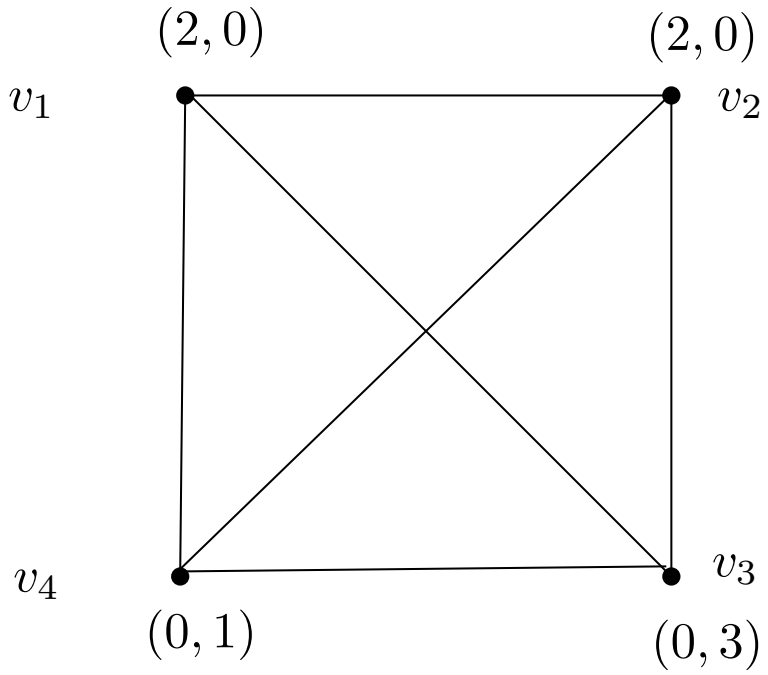

FIGURE 7. An example for which the optimal strategy requires the transportation of a bike several times between stations in default and stations in excess when $C = 3$, $p = v_1$ and $q = v_3$: the couples (x, y) near each vertex indicates the number x of bikes in the initial state and the number y of bikes in the terminal state.

If $\boldsymbol{z}$ is an admissible solution of program (5), we get

$$2\sum_{e\in E} z_e = \sum_{v\in V} z(\delta(v)) \geq 2\sum_{v\in V}\left\lceil \frac{|x_v - y_v|}{2}\right\rceil + \sum_{v\in U^{ex}} \eta(p,q,v) - \sum_{v\in U^{def}} \eta(p,q,v).$$

Since $\eta(p,q,v) = 0$ if $v \notin \{p,q\}$, $\eta(p,q,p) = -1$ and $\eta(p,q,q) = +1$, and assuming that p and q are unbalanced at the beginning of the algorithm, we have $\sum_{v\in U^{ex}} \eta(p,q,v) - \sum_{v\in U^{def}} \eta(p,q,v) = 2\eta(p,q,U^{ex})$ (just check all possibilities). Therefore

$$\sum_{v\in V}\left\lceil \frac{|x_v - y_v|}{2}\right\rceil + \eta(p,q,U^{ex})$$

is a lower bound for our problem. The conclusion is straightforward. □

The complexity status for the SSBP is open for a complete graph with unit costs and fixed $C \geq 3$ (this question is restated in Section 9). We conjecture that there is polynomial time algorithm.

If $C = 3$ an optimal solution can require the transportation of a bike several times between stations in default and stations in excess, a fact that does not occur when $C = 1$ or $C = 2$. Consider example in Figure 7. The optimal solution consists in the following sequence v_1, v_4, v_2, v_3, with 1 bike in the truck during the travel along edge v_4v_2.

8. A linear time algorithm in the case of a tree

8.1. The general case

In this section, we prove that the SSBP can be solved in polynomial time (actually linear time) in the case of a tree. Again, this means that we can compute the exact cost of an optimal balancing sequence, and not that the time to execute the sequence is polynomial. Nevertheless, the sequence can be described in a polynomial way, since we can write the optimal algorithm in this case.

We describe now the algorithm. Actually, we give an explicit solution to the research version of the SSBP in the case of a tree, that is, we give the optimal cost of an optimal sequence and the first move of an optimal sequence, which in a sense solves completely the problem since, as we have already noted, after a first move, we have again a problem of same nature.

Denote by $K_1, \ldots, K_h$ the connected components of $G - \{p\}$. Now, we will see that there is a rule depending only on the excess or default status of p and the h components that gives a first move of an optimal sequence.

We describe the rule.

(1) All stations are balanced
  (a) $p \neq q$
  move along the edge steming from p in direction of q.
  (b) $p = q$
  don't move (it is finished).
(2) There are unbalanced stations
  (a) There is a component K_i in excess
  take no bike and enter the component K_i in excess.
  (b) There is no component in excess
    (i) There are several components K_i such that we have simultaneously $x(K_i) \leq y(K_i)$ and at least one unbalanced station in K_i
    choose such a K_i such that $q \notin K_i$, take $\min(C, y(K_i) - x(K_i))$ bikes enter K_i and put the bikes on the first vertex of the component.
    (ii) There are only one component K_i such that we have simultaneously $x(K_i) \leq y(K_i)$ and at least one unbalanced station in K_i
    take $\min(C, y(K_i) - x(K_i))$ bikes enter K_i and put the bikes on the first vertex of the component.

We emphasize that to apply this rule, the truck must be empty. Since, after having applied it, the truck is again empty, we can apply it as many times as necessary to balance the whole tree.

In case 2b, the truck takes $\min(C, y(K_i) - x(K_i))$ bikes. In this case, it is always feasible since the truck is empty and since we have $x_p = y_p + \sum_{j=1}^{h} y(K_j) - \sum_{j=1}^{h} x(K_j) \geq y(K_i) - x(K_i)$ bikes on vertex p (no component is in excess).

Now define

- U_e denotes for an edge e the subset of vertices such that $\delta(U_e) = e$ and $x(U_e) \geq y(U_e)$ (if there is equality, make an arbitrary choice), and
- for each edge $e \in E$

$$z_e(p, q, \boldsymbol{x}, \boldsymbol{y}) := \max\left(2\left\lceil \frac{x(U_e) - y(U_e)}{C} \right\rceil + \eta(p, q, U_e), \mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{y})\right)$$

Theorem 5. *The rule described above provides a first move of an optimal sequence for the SSBP on a tree (and hence, if repeated, provides an optimal sequence) and the optimal cost is:*

$$\sum_{e \in E} c_e z_e(p, q, \boldsymbol{x}, \boldsymbol{y}). \tag{6}$$

Note that the rule is independent from the costs. This theorem solves the preemptive case. The non-preemptive case is **NP**-hard [12]. See Subsection 9.4 for further discussions on this point.

Proof of Theorem 5. Define

$$\gamma(p, q, \boldsymbol{x}, \boldsymbol{y}) := \sum_{e \in E} z_e(p, q, \boldsymbol{x}, \boldsymbol{y}).$$

We will first prove by induction on the value of $\gamma(p, q, \boldsymbol{x}, \boldsymbol{y})$ that each edge e is traversed exactly $z_e(p, q, \boldsymbol{x}, \boldsymbol{y})$ times when we apply the algorithm described above. Then according to Equation (2), it implies then that the sequence of moves is optimal.

If $\gamma(p, q, \boldsymbol{x}, \boldsymbol{y}) = 0$, then $\mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{y}) = 0$, $x(U_e) - y(U_e) = 0$ for each edge e, $p = q$ and all stations are balanced (for this second point, prove it by induction on the size of the tree, starting with a leave). Hence, we are in case 1b, and each edge is traversed exactly $z_e(p, q, \boldsymbol{x}, \boldsymbol{y}) = 0$ times, as required.

Now, suppose that $\gamma(p, q, \boldsymbol{x}, \boldsymbol{y}) > 0$. Let $(\boldsymbol{x}', p')$ be the state reached after the first move of the algorithm. Assume that we have proved that

$$z_e(p, q, \boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 + z_e(p', q, \boldsymbol{x}', \boldsymbol{y}) & \text{if } e = pp' \\ z_e(p', q, \boldsymbol{x}', \boldsymbol{y}) & \text{if not,} \end{cases} \tag{7}$$

Then, we have $\gamma(p', q, \boldsymbol{x}', \boldsymbol{y}) < \gamma(p, q, \boldsymbol{x}, \boldsymbol{y})$, whence induction can be applied for the state $(\boldsymbol{x}', p')$, aiming to reach $(\boldsymbol{q}, y)$. By induction, we know that the algorithm applied to $(\boldsymbol{x}', p')$ leads to a traversing of each edge e exactly $z_e(p', q, \boldsymbol{x}', \boldsymbol{y})$ times. Since the state $(\boldsymbol{x}', p')$ is obtained just after the move along edge pp', we get that each edge e is traversed $z_e(p, q, \boldsymbol{x}, \boldsymbol{y})$ times when we apply the algorithm. Hence, we only have to prove that (7) is correct, and it is what the remaining of the proof is devoted to.

Actually, it is easy to see that if $e \neq pp'$, then $x(U_e) = x'(U_e)$, $\eta(p, q, U_e) = \eta(p', q, U_e)$ and $\mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{y}) = \mu(p', q, U_e, \boldsymbol{x}', \boldsymbol{y})$. Hence, it is enough to prove Equation (7) only for $e = pp'$.

Case 1a

We have $z_e(p, q, \boldsymbol{x}, \boldsymbol{x}) = 1$ and $z_e(p', q, \boldsymbol{x}, \boldsymbol{x}) = 0$. Indeed $\eta(p, q, U_e) = \mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{x}) = 1$ and $\eta(p', q, U_e) = \mu(p', q, U_e, \boldsymbol{x}, \boldsymbol{x}) = 0$. Equation (7) is satisfied.

Case 2a

We have $U_e = K_i$. Note that after the move pp', the subset U_e remains the same since $x(U_e)$ remains greater than $y(U_e)$. If $q \in K_i$, we have $\eta(p, q, U_e) = 1$, $\eta(p', q, U_e) = 0$, $\mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{y}) = 1$ and $\mu(p', q, U_e, \boldsymbol{x}, \boldsymbol{y}) = 2$. Using the fact that $x(U_e) > y(U_e)$, we get that

$$z_e(p, q, \boldsymbol{x}, \boldsymbol{y}) = 2 \left\lceil \frac{x(U_e) - y(U_e)}{C} \right\rceil + \eta(p, q, U_e) = 1 + 2 \left\lceil \frac{x(U_e) - y(U_e)}{C} \right\rceil + \eta(p', q, U_e) = 1 + z_e(p', q, \boldsymbol{x}, \boldsymbol{y}),$$

as required. Hence Equation (7) is satisfied. If $q \notin K_i$, we have $\eta(p, q, U_e) = 0$, $\eta(p', q, U_e) = 1$, $\mu(p, q, U_e, \boldsymbol{x}, \boldsymbol{y}) = 2$ and $\mu(p', q, U_e, \boldsymbol{x}, \boldsymbol{y}) = 1$. Hence Equation (7) is satisfied for similar reasons.

Case 2(b)i

We have $\eta(p, q, \bar{K}_i) = 0$, $\eta(p', q, \bar{K}_i) = 1$, $\mu(p, q, K_i, \boldsymbol{x}, \boldsymbol{y}) = 2$ and $\mu(p', q, K_i, \boldsymbol{x}', \boldsymbol{y}) = 1$.

- If $y(K_i) - x(K_i) = 0$, Equation (7) is clearly satisfied.
- If $1 \leq y(K_i) - x(K_i) \leq C$: $U_e = \bar{K}_i$ before the move and we have

$$2 \left\lceil \frac{x(\bar{K}_i) - y(\bar{K}_i)}{C} \right\rceil + \eta(p, q, \bar{K}_i) = 2 \left\lceil \frac{y(K_i) - x(K_i)}{C} \right\rceil = 2$$

$$= 1 + 2 \left\lceil \frac{x(\bar{K}_i) - y(\bar{K}_i) - y(K_i) + x(K_i)}{C} \right\rceil + \eta(p', q, \bar{K}_i).$$

Hence Equation (7) is satisfied.

- If $y(K_i) - x(K_i) > C$: $U_e = \bar{K_i}$ before and after the move and we have

$$2\left\lceil \frac{x(U_e) - y(U_e)}{C} \right\rceil + \eta(p, q, U_e) = 1 + 2\left\lceil \frac{x(U_e) - y(U_e) - C}{C} \right\rceil + \eta(p', q, U_e).$$

  Thus Equation (7) is satisfied.

Case 2(b)ii

The study is similar to the previous case (with the additional checking whether q is in or not in K_i). □

8.2. The case of a line

In this case, the preemptive and non-preemptive version of the SSBP on the line are identical, at least if p is an endpoint of the line and q is not fixed.

Define $d_v := x_v - y_v$ and denote by $D(v_j) := \sum_{i=1}^{j} d_{v_i}$. To make the discussion simpler, we suppose that $d_v \leq C$ for all vertices v (it is easy to see that the algorithm described below can be adapted). If $\left\lceil \frac{D(v_i)}{C} \right\rceil$ and $\left\lceil \frac{D(v_{i+1})}{C} \right\rceil$ are different and in an increasing order, call v_i a *positive step*. If they are different and in a decreasing order, call v_i a *negative step*.

Now, start from p (assumed to be the left endpoint) and go to the right. If a positive step is followed by a negative step, it is possible to balance all stations strictly between them by a simple move from left to right. At the end of this move, the two steps have disappeared. Then, if there is still a positive step on the left of the truck, go back to the positive step, and then go again in the right direction till you reach a positive step followed by a negative step. Repeat these operations until all negative steps are on the left side and all the positive steps on the right side.

Now, if a negative step is followed by a positive step, it is possible to balance all stations strictly between them by a simple move from left to right till the positive step without moving any bike and then by coming back to the negative step, this time wile moving the bikes. At the end of this move, the two steps have disappeared. Repeat these operations until all stations are balanced.

By induction on the number of steps, we can prove the optimality of this algorithm.

In general, there may be a difference, even in the case of a tree. Indeed, consider the example of Figure 8. The cost $c(e)$ are all equal to 1 and the capacity of the truck is $C = 3$. We can balance all the stations in 8 moves. But we necessarily put bikes on station v and then later, we take them again in the truck and put them on stations w and q. If we want to avoid such situations, the minimal number of moves is 10.

9. Open questions

9.1. Circle

We have seen that the problem on a tree – and in particular on a line (see further properties of the line in Subsection 9.4) – is polynomial. What about the circle ? We could imagine stations on a circular route, around a park, for instance. The complexity status of this case is open.

9.2. Complete graph, unit costs and C fixed

We have seen that when $C = 1$ or $C = 2$ the problem is polynomial (Proposition 1). We conjecture that it remains polynomial for any fixed C. In particular, we conjecture that if $C = 3$, there is a polynomial algorithm that solves the SSBP problem on a complete graph with unit costs.

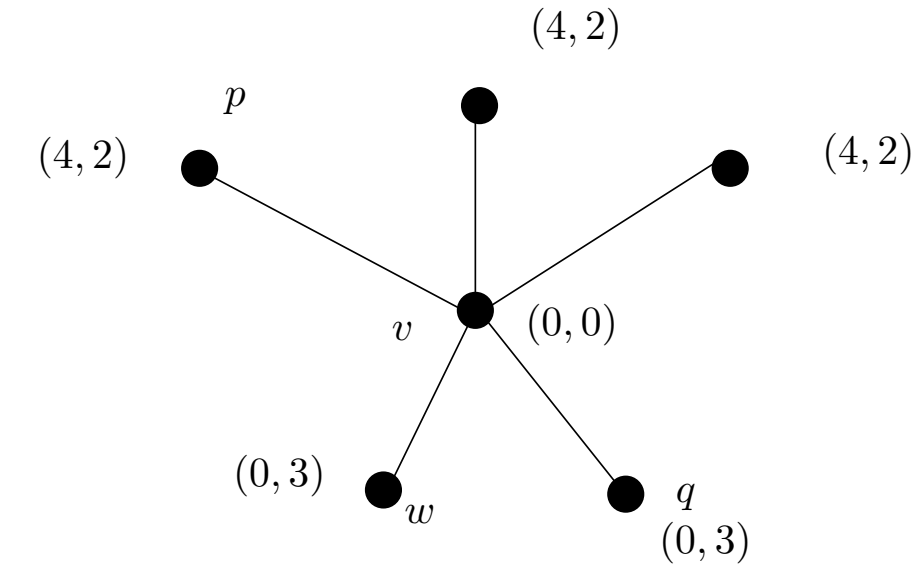

FIGURE 8. An exemple for which a cost for the load/unload operation is penalizing: the couples (x, y) near each vertex indicates the number x of bikes in the initial state and the number y of bikes in the terminal state.

9.3. Polynomial encoding of an optimal sequence of move

We have seen that it is not clear whether it is always possible to have a polynomial encoding of an optimal sequence balancing the stations. This can be an important issue not always because we want to be able to describe compactly an optimal strategy, but also if we want to explore the set of possible solutions (to implement a local search for instance).

9.4. A cost for the loading/unloading ?

In the same spirit of the distinction between the preemptive version and the non-preemptive one, we can consider the case when there is a cost for the load and the unload of bikes (proportional to the number of bikes for instance). In this case, we can try to optimize according both the distance and the total cost of loading and unloading. A more general model, and maybe more realistic, would include these issues.

As we have seen, in the particular case when the graph is a line, there is no difference but in general for a tree, there may be a gap.

9.5. Dynamic stations balancing

In the real situation, the bikes are moving between the stations while the truck tries to keep the stations balanced. We have justified the static version by the situation encountered at the end of the night, just before the morning rush, when there are few bike moves, but when there is a real need of a correct balanceness of the stations.

Nevertheless, there is a real need of a good heuristics in the dynamic case. Such heuristics could depend on the current situation (the number of bikes per stations is the only quantity known about the state of

the system at any instant – at least for the system designed in Paris), but also on the dynamic OD-matrix (which tells, in average, at any time of the day, how many bikes go from one station to another one).

References

1. S. Anily and J. Bramel, *Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries*, Naval Research Logistics (1997).
2. S. Anily and R. Hassin, *The swapping problem*, Networks **22** (1992), 419–433.
3. P. Augerat, J. M. Belenguer, E. Benavent, A. Corberán, and D. Naddef, *Seprating capacity constraints in the CRVP using tabu search*, European Journal of Operational Research **106** (1998), 546–557.
4. P. Chalasani and R. Motwani, *Approximating capacited routing and delivery problem*, SIAM Journal on Computing **28** (1999), 2133–2149.
5. M. Charikar, S. Khuller, and B. Raghavachari, *Algorithms for capacitated vehicle routing*, SIAM Journal on Computing **31** (2001), 665–682.
6. N. Christofides, *Worst-case analysis for a new heuristic for the Traveling Salesman problem*, Symposium on New Directions and Recent Results in Algorithms and Complexity (J.F. Traub, ed.), Academic Press, 1976.
7. M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completness*, W. H. Freeman, 1979.
8. H. Hernández-Pérz and J.-J. Salazar-González, *The one-commodity pickup-and-delivery travelling salesman problem*, Lecture Notes in Computer Science **2570** (2002), 89–104.
9. ———, *A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery*, Discrete Applied Mathematics **145** (2004), 126–139.
10. J.A. Hoogeveen, *Analysis of christofides'heuristic: some paths are more difficult than cycles*, Operations Research Letters **10** (1991), 291–295.
11. D. König, *Uber Graphen und ihre Anwendung auf Determinantentheorie une Mengenlehre*, Math. Ann. **77** (1916), 453–465.
12. A. Lim, F. Wang, and Z. Xu, *The capacitated traveling salesman problem with pickups and deliveries on a tree*, Lecture notes in Computer Science, Algorithms and Computation, Springer Berlin / Heidelberg, 2005.