The Multiple Vehicle Balancing Problem

Marco Casazza[*1,2], Alberto Ceselli[†1], Daniel Chemla[‡3,2], Frédéric Meunier[§3], and Roberto Wolfler Calvo[¶2]

[1]Università degli Studi di Milano, Dip. di Informatica, Via Bramante 65, Crema, Italy
[2]Université Paris 13, LIPN, 93430 Villetaneuse, France
[3]Université Paris Est, CERMICS, 77455 Marne-la-Vallée Cedex 2, France

December 15th, 2016

Abstract

This paper deals with the Multiple Vehicle Balancing Problem (MVBP). Given a fleet of vehicles of limited capacity, a set of stations with initial and target inventory levels and a distribution network, the MVBP requires to design a set of routes and pickup and delivery operations along each route such that inventory is redistributed among the stations without exceeding the vehicle capacities and such that routing costs are minimized. The MVBP arises in bicycle sharing systems, where rebalancing of bicycles is needed between stations to match their expected demands. The MVBP is NP-hard, generalizing several problems in transportation like the Split Delivery Vehicle Routing Problem. Using theoretical properties of the problem, we propose an integer linear programming formulation. Lower bounds are computed by a column generation routine embedding an ad-hoc pricing algorithm; we also introduce strengthening valid inequalities. Upper bounds are obtained by a memetic algorithm that focuses on routing and considers the pickup and delivery operations in a post-processing phase. We combine lower and upper bounding routines in both exact and matheuristic algorithms, obtaining proven optimal solutions for MVBP instances with up to 25 stations and an unbounded number of vehicles, or up to 20 stations and 5 vehicles.

Keywords: Bicycle sharing system, column generation, dominance properties, memetic algorithm, valid inequalities, vehicle routing

---

[*]Corresponding Author: marco.casazza@unimi.it

[†]alberto.ceselli@unimi.it

[‡]daniel.chemla@gmail.com

[§]frederic.meunier@enpc.fr

[¶]roberto.wolfler@lipn.univ-paris13.fr

1   Introduction

Context

Self-service bicycle sharing systems are now widespread all over the world. After several tries in different countries, systems like Vélo'V in Lyon (France) launched in 2005 and its follower in Paris Vélib two years later have encountered a remarkable success. Bicycle sharing systems are now present in more than 200 cities, including New York (U.S.A.) and Hangzhou (P.R.C.), the biggest bicycle sharing network being in the latter city with about $2'400$ stations and up to $60'000$ bicycles. One of the issues faced by operators is to ensure that users are consistently able to find bicycles at their departure station and available drop racks at their destination station. In fact, especially during peak hours, many users choose the same departure and arrival stations, yielding bicycles unavailability at the first stations and empty racks unavailability at the latter stations. The solution chosen by many operators is to rebalance stations by means of dedicated trucks, which pickup bicycles from certain stations and deliver them to other ones, in order to fulfill at best an estimated demand. Given the costs and the constraints of driving trucks in a urban environment, being able to efficiently perform such a rebalancing is a key factor for the success of the whole system. There are many other issues, e.g. station locations, rebalancing incentives, fleet dimensioning. The interested reader can find in Laporte et al. [17] an updated survey on the operations research questions raised by bicycle sharing systems. The topic of the present paper comes under the first mentioned issue, namely the rebalancing problem. More precisely, we are interested by *static* rebalancing performed by several trucks, where "static" means that no users are allowed to move the bicycles. This is the case when the system is closed or nearly idle during night. Static rebalancing is classically opposed to *dynamic* rebalancing, where the moves performed by the users cannot be neglected. There is currently a lively research trend on these static and dynamic rebalancing problems. Raviv et al. [20] wrote the first paper addressing the static problem with more than one truck. They propose several models whose objective is to maximize user satisfaction, and formulate them as mixed integer programs solved by CPLEX. Since this approach is computationally expensive and considerably restricts the size of the instances that can be solved within reasonable time, the authors propose also a two-phase heuristic. Forma et al. [15] later presented a three-step matheuristic for the same problem, which outperforms previous algorithms. Schuijbroek et al. [22] propose a model inspired by [20], but different in the way of calculating the users satisfaction (they use a server queueing system point of view), which is solved through a cluster first-route second heuristic yielding high quality solutions on real instances. Dell'Amico et al. [11] considered a problem close to ours, with the objective of minimizing the total routing cost. Nevertheless, they assume each station served by a single truck. They propose four mixed integer linear programming formulations for the problem, solved by a branch-and-cut able to optimally solve all their instances (taken from real systems) involving up to 50 stations and obtained relatively low optimality gaps

in most of the remaining cases (up to 116 stations). We refer to the survey mentioned above for other references on rebalancing.

The Multiple Vehicle Balancing Problem

In this paper we introduce a new routing problem, capturing the main features of the task faced by the fleet of trucks in a static rebalancing context. We call it the Multiple Vehicle Balancing Problem (MVBP). We assume that a homogeneous fleet of vehicles of limited capacity is given, together with a set of stations with initial and target inventory levels. The initial level is the number of bicycles originally present in the station while the target level represents the number of bicycles needed to meet an expected demand. We also assume that the distances between the stations are known. The MVBP requires to find a set of routes, each composed by a sequence of stations to be visited by a vehicle and pickup and delivery operations along the route. The objective function is the total distance travelled by the vehicles. Temporary storage is not allowed, i.e. no bicycle can be loaded from a station whose target inventory is higher than or equal to the initial one, or unloaded to a station whose target inventory is lower than or equal to the initial one. In fact, temporary storage is uncommon in real systems, since they may lead to vehicle synchronization issues. In Section 2.1, we give the precise description of the problem.

From a methodological point of view, the MVBP is NP-hard and generalizes several routing problems. For instance, when all stations have a target inventory that is higher than the initial one, except for a single depot station, our MVBP becomes the Split Delivery Vehicle Routing Problem (SDVRP) [3]. Dror and Trudeau [13] have been the firsts to study this problem. They prove that the split dimension of the problem may lead to savings in term of the objective function; at the same time this feature substantially increases the complexity of the problem [3]. Archetti *et al.* [4] present a tabu search algorithm to solve this problem. A few years later, Archetti *et al.* [2] and Archetti *et al.* [1] propose exact algorithms exploiting column generation and cutting planes, respectively. They manage to find proven optimal solutions for instances with up to 100 customers, and one with 144 vertices, in a few hours of computation.

The MVBP belongs to the wide class of *Pickup and Delivery vehicle routing problems* (PDVRPs), where a fleet of vehicles is used to transport requests from the depot and/or some vertices to the depot and/or other vertices in the network. For recent survey on PDVRPs, we refer the interested reader to Baldacci *et al.* [6] for what concerns transportation of freight, and Schmid *et al.* [21] for transportation of people. There is a vast literature on the PDVRP, but to discuss this huge literature is out of the scope of the paper. Nevertheless we cite Baldacci *et al.* [5], since the authors propose one of the most powerful exact approaches for the PDVRP with time windows (PDPTW).

The MVBP can be classified as a *many-to-many* (M-M) vehicle routing problem, in which a request has multiple origins (in our case pickup stations) and multiple destinations (in our case delivery stations). It can also be seen as the multi-vehicle counterpart of single vehicle problem studied by the three

last authors in [9], even if in this latter work – and it is one of its interests – temporary storage is allowed.

Contribution and plan

Our main contributions are the following:

- we investigate distinctive theoretical properties of optimal solutions. Such properties extend those by Dror and Trudeau for the SDVRP [13];

- with the help of the aforementioned extensions of Dror and Trudeaus properties we model the MVBP with a set partitioning extended formulation having one variable for each route with pickup and delivery operations;

- we design an effective way to solve the relaxation of our formulation by means of column generation techniques, solving the pricing problem with an algorithm adapted from Baldacci et al. [7];

- we introduce additional cuts for the MVBP to enhance the quality of the lower bound and we show how to integrate them efficiently with our pricing procedure;

- we present how to produce tight upper bounds by means of an efficient memetic algorithm exploiting a particular combinatorial encoding of the solutions. Such matheuristic solves to optimality almost all instances with up to 20 vertices in an average computing time smaller than 10 minutes;

- we integrate together the procedures computing the lower and upper bounds into an optimization framework that solves to optimality instances with up to 20 vertices.

The rest of the paper is organized as follows: in Section 2 we focus on the definition of the Multiple Vehicle Balancing Problem (MVBP) and of its main properties. In Section 2.3 we show distinctive properties of the optimal solutions and we model the MVBP in Section 2.4. Then, in Section 3 we show how to obtain feasible lower bounds and we also detail our pricing problem. In Section 4 we introduce additional cuts. In Section 5, we describe a special polynomially solvable subproblem and our memetic algorithm. In Section 6 we present our optimization framework. Computational results and conclusions are given in Section 7 and Section 8, respectively.

2 Models and properties

2.1 Problem formulation

We are given a complete directed graph $G = (V, A)$ without loops. The vertex set V is identified with integers $0, 1, \ldots, n$, where 0 is a special vertex that can be seen as the depot. For each arc (i, j), there is a positive cost c_{ij}. These costs

are assumed to satisfy the triangular inequality. There are M vehicles, each of them of capacity $Q \in \mathbb{Z}_+$, aiming at moving inventory units in the graph. An inventory level $d_i \in \mathbb{Z}$ is attached to each vertex i. If $d_i > 0$, there is a *supply* of d_i units at i, which is then a *pickup vertex*. If $d_i < 0$, there is a *demand* of d_i units at i, which is then a *delivery vertex*. The following relations are assumed to hold throughout the paper:

$$\sum_{i=0}^{n} d_i = 0, \qquad |d_i| \leq Q \quad \forall i \in V, \qquad \text{and} \qquad d_0 = 0. \qquad (1)$$

The assumption $|d_i| \leq Q$ is not a strong restriction: a vertex i violating such inequality can always be replaced by a clique of vertices $i^1, i^2, \ldots, i^k$ with $k = \lceil |d_i|/Q \rceil$ such that their inventory levels are equal to Q, except for i^k for which we have $|d_{i^k}| = |d_i| - Q \cdot (k-1)$. The vehicles are requested to follow routes, starting and ending at vertex 0, along which they carry inventory from the pickup vertices to the delivery vertices in order to fulfill exactly the demand. The vertices can be visited several times and by distinct vehicles, but temporary storage is not allowed. It means that when inventory units are delivered at some vertices, they cannot be picked up later again. It also means that during the carrying process, the inventory level of each station varies monotonically, and that a vertex i with $d_i = 0$ is never visited, except when $i = 0$.

Moreover, we call *stop* each vertex visit of a route and we assume that we are given a bound T on the number of stops in a route. In such a way we distribute the vertex visits between vehicles imposing a level of fairness among workers.

A *schedule* of a vehicle is a route with the load of the vehicle along each arc traversed by the route. The cost of a schedule is the sum of the costs of the arcs traversed by its route, and the cost of a collection of schedules is the sum of the costs of the schedules. The *Multiple Vehicle Balancing Problem* aims at finding the vehicle schedules satisfying the above requirements at a minimal cost.

2.2 Practical meaning

The graph G models the shortest paths in the bicycle sharing system. Each vertex apart 0 is a station and the inventory is made of bicycles. The vehicles are the trucks used for the rebalancing. A slightly more general problem would allow temporary storage at stations – this is for instance the case in the already cited work [9] – but the problem would then be of a formidable complexity, especially because it would require synchronization issues. Since temporary storage is not permitted, only the algebraic difference d_i between initial and target levels matters. The bound T is a way to distribute the tasks between the vehicles. If $T = +\infty$, the optimal solution is achieved by a single vehicle: concatenating the schedules of a feasible solution provides a solution of same cost for a single vehicle, and this solution can be in general made smaller because of the triangular inequality.

2.3 Profitability properties

The MVBP have many degrees of freedom, leaving the option of combining schedules in different ways. However, the following *profitability* properties hold, which will allow us to consider a restricted set of solutions.

Consider the following features a solution of the MVBP may have.

Feature 1. *Given any pair $\{i,j\}$ of pickup or delivery vertices, the total number of traversals of arcs (i,j) and (j,i) is at most one over all schedules of the solution.*

Feature 2. *Given any pickup vertex i and any delivery vertex j, the arc (i,j) is traversed at most once with a load smaller than Q over all schedules of the solution, and the arc (j,i) is traversed at most once with a load larger than 0 over all schedules of the solution.*

Feature 3. *Given any pickup vertex i and any delivery vertex j, if there is at least one traversal of (i,j) and one traversal of (j,i) in one schedule of the solution, then all traversals of (i,j) of all schedules of the solution are done with a load of Q inventory units.*
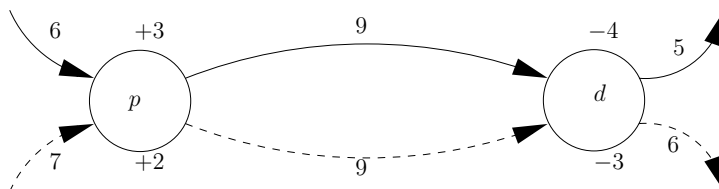

Figure 1: Assume $Q = 10$. The solid line route and the dotted line route correspond both to a non-full vehicle traveling the arc (p, d). Transferring one picked-up inventory unit from one of the route to the other leads to a move between p and d with 10 inventory units – which is the vehicle capacity – and the other with 8 inventory units, without changing the impact on the vertices.

The following proposition is a generalization of the Dror and Trudeau dominance property [13].

Proposition 1. *Provided that the problem is feasible, there always exists an optimal solution satisfying simultaneously Features 1, 2, and 3.*

Proof. The proof consists in showing that given an optimal solution, one can always perform local changes without affecting neither the feasibility nor the optimality of the schedules, and get a solution satisfying all Features simultaneously.

Feature 1 Consider an optimal solution. There is a *traversal of $\{i,j\}$* each time there is a traversal of either (i,j) or (j,i) in a route. Suppose that we have

an optimal solution with at least two traversals of a pair $\{i, j\}$ of pickup vertices. Denote a_1 (resp. a_2) the number of inventory units loaded at i during the first (resp. second) traversal, and denote b_1 (resp. b_2) the number of inventory units loaded at j during the first (resp. second) traversal. Two cases have to be considered.

Case $a_2 \geq b_1$**.** We can take $a_1 + b_1$ inventory units at i instead of a_1 and skip j after the first traversal. The solution remains unchanged except for the second traversal, where we take $a_2 - b_1$ inventory units on i and $b_1 + b_2$ inventory units on j.

Case $a_2 < b_1$**.** We can take $a_1 + a_2$ inventory units at i instead of a_1 and take $b_1 - a_2$ inventory units at j during the first traversal. The solution remains unchanged except for the second traversal, where we skip i and go directly to j, loading $a_2 + b_2$ inventory units.

In both cases, we are able to reduce by one the number of traversals of $\{i, j\}$ without further changing the solution, thereby without increasing its cost. Such a reduction can be repeated if more than two traversals are performed. The proof is similar if both i and j are delivery vertices.

Feature 2 Consider an optimal solution satisfying Feature 1. Let i be a pickup vertex and j be a delivery one. Assume that the solution traverses at least twice the arc (i, j), each time with strictly less than Q inventory units. Transferring inventory units between these two traversals either allows to save one traversal, or provides a traversal with Q inventory units. See Figure 1 for an illustration. We get a similar conclusion if i is a delivery vertex and j a pickup one.

Feature 3 Consider an optimal solution satisfying Features 1 and 2. Let i be a pickup vertex and j be a delivery one. Suppose that there is both a traversal of (i, j) with strictly less than Q inventory units, and a traversal of (j, i). We assume that during the traversal (j, i), there is at least one inventory unit delivered on j and one inventory unit picked up on i, as otherwise one vertex can be skipped without increasing the solution cost. This inventory unit could be picked up from i and delivered on j during the unsaturated traversal of (i, j). Hence, we can repeat this process until there is a load of Q inventory units during the traversal of (i, j). □

Proposition 1 ensures, as a corollary, that every schedule of an optimal solution satisfies independently those three features. In our algorithms we will thus consider them in the following single schedule form.

Feature 1s. *Given any pair $\{i, j\}$ of pickup or delivery vertices, the total number of traversals of arcs (i, j) and (j, i) in the schedule is at most one.*

Feature 2s. *Given any pickup vertex i and any delivery vertex j, the arc (i, j) is traversed at most once with a load smaller than Q by the schedule, and the arc (j, i) is traversed at most once with a load larger than 0 by the schedule.*

Feature 3s. *Given any pickup vertex i and any delivery vertex j, if there is at least one traversal of (i,j) and one traversal of (j,i) in the schedule, then all traversals of (i,j) by the schedule are done with a load of Q inventory units.*

2.4 Integer programming model

For a schedule s, we denote by $l_i^s \in \mathbb{Z}_+$ the total number of inventory units either picked up or delivered at i, and by c^s its cost. A schedule s is *feasible* if the following hold simultaneously:

- its route starts and finishes at vertex 0, and contains at most T stops $i \neq 0$,

- the number of inventory units carried over each arc of the route is at most Q,

- we have $1 \le l_i^s \le |d_i|$ for all $i \in V \setminus \{0\}$,

Let $\mathcal{S}$ be the set of feasible schedules that satisfy moreover Features 1s, 2s, and 3s. According to Proposition 1, the following integer program models the MVBP.

$$
\begin{aligned}
\min \quad & \sum_{s \in \mathcal{S}} c^s z^s \\
\text{s.t.} \quad & \sum_{s \in \mathcal{S}} l_i^s z^s = |d_i| \qquad \forall i \in V \quad \text{(i)} \\
& \sum_{s \in \mathcal{S}} z^s \le M \qquad\qquad \text{(ii)} \\
& z^s \in \{0,1\} \qquad\quad \forall s \in \mathcal{S}.
\end{aligned}
\tag{P}
$$

Nothing prevents a priori to choose several times the same schedule for different vehicles. However, since we suppose that $|d_i| \le Q$ for each vertex i, any optimal solution satisfying Proposition 1 selects each possible schedule at most once. Indeed, Feature 2 ensures that if the same schedule is used by two vehicles or more, its route is a sequence that alternates between pickup and delivery vertices, and the load of the vehicle is Q when leaving a pickup vertex and 0 when leaving a delivery vertex. The fact that $|d_i| \le Q$ for all i prevents then the assignment of a same schedule to two vehicles or more. We therefore restrict the values of z^s to the set $\{0,1\}$.

3 Continuous relaxation

In this section, we describe a method to solve the continuous relaxation of (P), which we call (RP). We adopt a column generation approach, for which we assume basic knowledge (see [12] for an introduction). We focus in this section on how to solve the *pricing problem*, which consists here in finding the schedule

$s \in \mathcal{S}$ of minimal reduced cost

$$\bar{c}^s = c^s - \sum_{i \in V} \lambda_i l_i^s - \nu, \tag{2}$$

where the $\lambda_i \in \mathbb{R}$ and $\nu \in \mathbb{R}_-$ are respectively the values taken by the dual variables associated to constraints (i) and constraint (ii) in (RP). The pricing problem turns out to be a Shortest Path Problem with Resource Constraints (see for example [16]). In our case paths do not need to be elementary: multiple visits to the same vertex, each partially serving its demand, are allowed. The method we propose is inspired by that of Baldacci *et al.* [5] for the PDPTW. The procedure is divided into two phases: GENPATH and GENROUTE. In GENPATH, *partial schedules* are computed. These partial schedules are either *forward partial schedules* or *backward partial schedules*, which are respectively starting at the depot and ending at an arbitrary vertex or vice versa. These partial schedules are built by a systematic enumeration procedure, but the enumeration is restricted only to those partial schedules that can be extended to schedules in $\mathcal{S}$. In addition, dominance properties are used. In GENROUTE, forward and backward partial schedules are selectively joined in feasible schedules.

3.1 The GENPATH-GENROUTE procedure

Several adaptations are required to the algorithm of Baldacci *et al.* [5] for fitting the MVBP, as the number of inventory units to be picked up or delivered at each stop has to be decided, the vertices may be visited several times, and pickup and delivery operations are not paired.

We define as *forward partial schedule P* a portion of a schedule in $\mathcal{S}$, starting at the depot and ending at an arbitrary vertex $e(P)$ with a load $\ell(P)$, including at most $\lfloor \frac{T}{2} \rfloor$ stops. A *backward partial schedule* is defined similarly, representing a portion of a schedule in $\mathcal{S}$ starting at an arbitrary vertex $e(P)$ with a load $\ell(P)$, and ending at the depot, including at most $\lceil \frac{T}{2} \rceil$ stops. To ease notation we denote by r_k^P the vertex that is the kth stop in the partial schedule P, and by l_i^P the total number of inventory units picked up or delivered at vertex i in partial schedule P. The *cost of a partial schedule P* with t stops is defined as

$$\bar{c}_P = \sum_{k=1}^{t} (c_{r_k^P r_{k+1}^P} - \lambda_{r_k^P} l_{r_k^P}).$$

The generation of the set of forward partial schedules $\overrightarrow{\mathscr{P}}$ in GENPATH is detailed in Algorithm 1. In Step 13, by "P' violates feasibility rules", we mean "$l_i^{P'} > d_i$ or the load of the vehicle exceeds Q on some arc", and by "P' violates profitability properties", we mean that at least one of Features 1s, 2s, and 3s is not satisfied by P'.

GENPATH exploits partial enumeration and dominance tests to generate the set of partial schedules that can belong to schedules in $\mathcal{S}$. As in a classic

Algorithm 1 GENPATH for forward partial schedules

1: Create the empty partial schedule P_0 with $\bar{c}_{P_0} = 0, e(P_0) = 0, l_i^{P_0} = 0$ for each $i \in V$, $\mathscr{P} = \{P_0\}$ and $\overrightarrow{\mathscr{P}} = \emptyset$

2: **if** $\mathscr{P} = \emptyset$ **then**

3:    STOP

4: **end if**

5: $P^* = \arg\min\{lb(P), P \in \mathscr{P}\}$

6: $\mathscr{P} = \mathscr{P} \setminus \{P^*\}$

7: Insert in $\overrightarrow{\mathscr{P}}$ forward partial schedule P^*

8: **if** $|P^*| = \left\lfloor \frac{T}{2} \right\rfloor$ **then**

9:    Return to 2

10: **end if**

11: **for** $i \in V$ and $\delta \in \{0, \ldots, d_i - l_{e(P^*)}^{P^*}\}$ **do**

12:    Expand P^* from $e(P^*)$ to vertex i, obtaining a new partial schedule P', and setting $\ell(P') = \ell(P^*) + \delta$ (if i is pickup) or $\ell(P') = \ell(P^*) - \delta$ (if i is delivery), $l_j^{P'} = l_j^{P^*}$ for each $j \in V \setminus \{i\}$ and $l_i^{P'} = l_i^{P^*} + \delta$

13:    **if** P' violates feasibility rules or profitability properties **then**

14:       Reject P'

15:       Return to 2

16:    **end if**

17:    let $\bar{c}_{P'} = \bar{c}_{P^*} + c_{e(P^*)i} - \lambda_i \delta$

18:    Compute $lb(P')$

19:    **if** $lb(P') > \rho$ **then**

20:       Reject P'

21:       Return to 2

22:    **end if**

23:    **if** P' is dominated by any partial schedule in $\mathscr{P}$ **then**

24:       Reject P'

25:       Return to 2

26:    **end if**

27:    Remove all partial schedules in $\mathscr{P}$ dominated by P'

28:    Insert P' in $\mathscr{P}$

29: **end for**

30: Return to 2

---

column generation approach, we select schedules having negative reduced cost. However for the sake of the exposition we use an additional parameter ρ acting as a threshold: during the pricing phase, only schedules having a reduced cost less or equal to ρ are selected, ρ being equal to 0 when we compute the optimal value of (RP). Such a threshold provides the additional flexibility that is needed for a subsequent enumeration step, detailed in Section 6. A set of potentially useful partial schedules $\mathscr{P}$ is kept, which initially contains a single empty partial schedule. A *lower bound $lb(P)$* on the minimum cost a schedule completing P is computed (see Section 3.2). Iteratively, the partial schedule in $\mathscr{P}$ having best bound is selected and extended, creating a new partial schedule for each pickup (resp. delivery) vertex i and pickup (resp. delivery) inventory value δ that do not violate feasibility rules and profitability properties; new partial schedules are rejected also if the corresponding bound exceeds ρ. Finally, dominance rules are applied to further reduce the number of partial schedules in $\mathscr{P}$ (see Section 3.3). GENPATH is run twice, to obtain both forward and backward partial schedules.

GENROUTE is then run to selectively join the generated partial schedules into a final set of all feasible, profitable, and non-dominated schedules whose reduced cost is at most ρ. When sets $\overrightarrow{\mathscr{P}}$ of forward partial schedules and $\overleftarrow{\mathscr{P}}$ of backward partial schedules are generated, we create full schedules by joining their elements. We partition each of these sets into subsets $\overrightarrow{\mathscr{P}}_{iw}$ and $\overleftarrow{\mathscr{P}}_{iw}$ of partial schedules P with $\ell(P) = w$ and $e(P) = i$. The procedure is explained in detail in Algorithm 2.

Algorithm 2 GENROUTE

1: $\mathcal{S} = \emptyset$
2: **for** $i \in V$ and $w \in \{0, \dots, Q\}$ **do**
3:    **for** $P \in \overrightarrow{\mathscr{P}}_{iw}$ and $P' \in \overleftarrow{\mathscr{P}}_{iw}$ **do**
4:       **if** P and P' are compatible **then**
5:          $s \leftarrow \text{append}(P, P')$
6:          **if** s is not dominated by any schedule in $\mathcal{S}$ **then**
7:             Add s to $\mathcal{S}$
8:             Delete from $\mathcal{S}$ all schedules dominated by s
9:          **end if**
10:       **end if**
11:    **end for**
12: **end for**

A forward partial schedule P and a backward partial schedule P' are *compatible* (Step 4) if the schedule obtained by appending P' to P respects all feasibility rules and profitability conditions described above for the GENPATH procedure. The schedule s obtained by combining P and P' has reduced cost $\bar{c}^s = \bar{c}^P + \bar{c}^{P'} - \nu$. A schedule s *dominates* a schedule s' (Steps 6 and 8) if $l_i^s = l_i^{s'}$ for all $i \in V$ and $\bar{c}^s \leq \bar{c}^{s'}$. To increase the speed of the algorithm, we sort $\overrightarrow{\mathcal{P}}_{iw}$ and $\overleftarrow{\mathcal{P}}_{iw}$ by non-increasing costs: as soon as $\bar{c}^P + \bar{c}^{P'} - \nu > \rho$, inner loops can be stopped, directly jumping to the next iteration of the outer ones.

3.2 Completion bound

We briefly explain how the lower bound $lb(P)$ on the minimum reduced cost of a schedule completing P is computed. We actually use a standard trick in a vehicle routing context to bound the cost of a solution: we compute the partial schedule by dynamic programming in such a way that each time we extend to a vertex, its inventory level at previous steps is neglected. Computing the minimum cost $\overrightarrow{\pi}(i, w)$ of a forward partial schedule ending at node i with final load w or the minimum cost $\overleftarrow{\pi}(i, w)$ of a backward partial schedule starting at node i with initial load w can then be solved in polynomial time by dynamic programming.

For P a forward partial schedule, we set $lb(P) = \bar{c}^P + \overrightarrow{\pi}(e(P), \ell(P)) - \nu$ and for P a backward partial schedule, we set $lb(P) = \bar{c}^P + \overleftarrow{\pi}(e(P), \ell(P)) - \nu$.

3.3 Dominance

Let P and P' be two forward partial schedules. The partial schedule P *dominates* the partial schedule P' if

$$e(P) = e(P'), \quad \ell(P) = \ell(P'), \quad l_i^P \le l_i^{P'} \text{ for all } i \in V, \quad \text{and} \quad \bar{c}_P \le \bar{c}_{P'} - \rho.$$

Indeed, when these dominance conditions are satisfied, any backward partial schedule B that is compatible with P' is also compatible with P, and therefore combining B with P always yields a better schedule. Moreover, even if combining B with P produces an optimal schedule, no schedule can be produced by combining B with P' whose value is within a constant ρ from optimality. Therefore, partial schedule P' can be rejected without affecting optimality guarantees.

It is also useful to directly prevent the following set of extensions always yielding dominated partial schedules. Let $r \to p \to d \to s$ be a sequence of consecutive vertices in a partial schedule, where p is a pickup vertex from where w bicycles are picked up and d is a delivery one where w' are delivered. An improved partial schedule skipping either p, d or both can be obtained in following cases:

if $(w < w')$ **and** $(c_{r,p} + w(\lambda_p + \lambda_d) - (c_{r,p} + c_{p,d}) < -\rho)$ the forward partial schedule skipping p and delivering only $(w' - w)$ bicycles at d has a lower cost

if $(w = w')$ **and** $(c_{rs} + w(\lambda_p + \lambda_d) - (c_{rp} + c_{pd} + c_{ds}) < -\rho)$ the forward partial schedule skipping both p and d has a lower cost

if $(w > w')$ **and** $(c_{ps} + w'(\lambda_p + \lambda_d) - (c_{pd} + c_{ds}) < -\rho)$ the forward partial schedule skipping d and picking up only $(w - w')$ bicycles at p has a lower cost

Similar conclusions can be drawn when the sequence is $r \to d \to p \to s$, where d is a delivery vertex and p is a pickup one (see Figure 2).
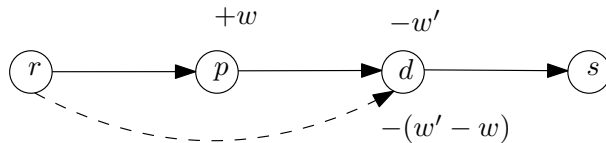
12

Figure 2: New forward partial schedule when $w < w'$ and when the cost of the forward partial schedule skipping p is lower than the one of the original

3.4 Implementation

Here, we provide details on how the pricing subproblem is solved in practice, for which ρ is set to 0. Actually, we do not go in general to the end of the GENPATH-GENROUTE procedure and thus we do not compute the exact value of $\min_{s \in \mathcal{S}} \bar{c}^s$. We stop GENPATH as soon as $15'000$ partial schedules have been generated[1]; if GENROUTE is able to find schedules whose reduced cost is negative, we perform early termination, which means inserting the corresponding columns in the current pool of columns and going back to the master problem. Otherwise, we resume GENPATH, generate up to $100'000$ partial schedules, and call GENROUTE again; as before, if negative reduced cost schedules are found we perform early pricing termination. Otherwise, we go to the end of the GENPATH-GENROUTE procedure to obtain the true value of $\min_{s \in \mathcal{S}} \bar{c}^s$.

4 Valid inequalities

Two families of cuts are added to improve the quality of the lower bound. First, a family of *dual-feasibility cuts* is obtained through dual-feasible functions. Second, a set of *profitability cuts* is derived from the properties of Section 2.3.

We discuss for each family the corresponding impact on the pricing algorithms. According to preliminary experiments, we found it useful to directly insert all these cuts in the initial (RP), i.e. no dynamic cut separation is performed in our implementation.

4.1 Superadditive and dual-feasible cuts

We consider three types of cuts obtained by applying some special families of functions on both sides of constraints (i) of (P).

The first family is a novel one, which we specially designed for our MVBP. Let k and d be two positive integers such that $k \leq d$. We define for $x \in [0, d]$ the mapping

$$F_{k,d}(x) := \begin{cases} 2\frac{k}{d}x - 1 & \text{if } \frac{k}{d}x \text{ is a positive integer} \\ 2\left\lfloor \frac{k}{d}x \right\rfloor & \text{otherwise.} \end{cases}$$

---

[1]or as soon as all possible partial schedules have been generated, in case of small instances.

Next proposition states that the functions $F_{k,d}$ are *superadditive*.

Proposition 2. $F_{k,d}$ *satisfies the following*

$$F_{k,d}(x) + F_{k,d}(y) \le F_{k,d}(x+y)$$

for all $x, y \in [0, d]$ *such that* $x + y \le d$

Proof. When $\frac{k}{d}(x+y)$ is not an integer, we have $F_{k,d}(x) + F_{k,d}(y) \le 2\lfloor \frac{k}{d}x \rfloor + 2\lfloor \frac{k}{d}y \rfloor \le 2\lfloor \frac{k}{d}(x+y) \rfloor = F_{k,d}(x+y)$. When both $\frac{k}{d}x$ and $\frac{k}{d}y$ are integer, we have $F_{k,d}(x) + F_{k,d}(y) = F_{k,d}(x+y)$. To finish the proof, we consider the only remaining case, namely when $\frac{k}{d}(x+y)$ is an integer while none of $\frac{k}{d}x$ and $\frac{k}{d}y$ are integer. Then, an integer $\jmath$ exists such that $\frac{k}{d}x + \frac{k}{d}y = \jmath$, and since none of the two summands on the left hand side is integer, we have $\lfloor \frac{k}{d}x \rfloor + \lfloor \frac{k}{d}y \rfloor \le \jmath - 1$. Hence $F_{k,d}(x) + F_{k,d}(y) = 2\lfloor \frac{k}{d}x \rfloor + 2\lfloor \frac{k}{d}y \rfloor \le 2\jmath - 2 = F_{k,d}(x+y) - 1 \le F_{k,d}(x+y)$. $\square$

We also considered two families of functions from the literature: $F_{k,d}^{FS,1}$ and $F_{k,d}^{CCM,1}$, introduced respectively by Fekete and Schepers [14] and by Carlier *et al.* [8]. They are defined as follows for $k \le d$ and $x \in [0, d]$:

$$F_{k,d}^{FS,1}(x) := \begin{cases} kx & \text{if } \frac{(k+1)x}{d} \text{ is integer} \\ \lfloor \frac{(k+1)x}{d} \rfloor d & \text{otherwise.} \end{cases}$$

$$F_{k,d}^{CCM,1}(x) := \begin{cases} 2(\lfloor \frac{d}{k} \rfloor - \lfloor \frac{d-x}{k} \rfloor) & \text{if } x > \frac{d}{2} \\ \lfloor \frac{d}{k} \rfloor & \text{if } x = \frac{d}{2} \\ 2\lfloor \frac{x}{k} \rfloor & \text{if } x < \frac{d}{2}. \end{cases}$$

$F_{k,d}^{FS,1}$ has been proved in [14] to be superadditive. The map $F_{k,d}^{CCM,1}$ is not superadditive, but Carlier *et al.* (Theorem 4.3 in [8]) proved that it is *dual-feasible*, meaning that

$$\sum_j F_{k,d}^{CCM,1}(x_j) \le F_{k,d}^{CCM,1}(d)$$

holds for any list of $x_j \in [0, d]$ such that $\sum_j x_j \le d$. We refer to [10] for a complete survey on this technique.

Applying a superadditive or dual-feasible map f on constraints (i) of (P) leads to new valid inequalities

$$\sum_{s \in \mathscr{R}} f(l_i^s) z^s \le f(|d_i|).$$

With the maps $F_{k,|d_i|}$, $F_{k,|d_i|}^{CCM,1}$, and $F_{k,|d_i|}^{FS,1}$, we get thus several new valid inequalities, for any choice of indices k. The reduced cost of a schedule (2) becomes then

$$c^s - \sum_{i \in V} \left(\lambda_i l_i^s + \sum_k (\mu_{k,i} F_{k,|d_i|}(l_i^s) + \eta_{k,i} F_{k,|d_i|}^{FS,1}(l_i^s) + \zeta_{k,i} F_{k,|d_i|}^{CCM,1}(l_i^s)) \right) - v,$$

where μ, η, and ζ are the dual variables, and where the sum over k is made over the chosen indices.

The GENPATH-GENROUTE procedure works with almost no modifications for these new reduced costs. When the algorithm extends to a node i with a current load $l_i^{P^*}$ and loading δ inventory units, the cost of the partial schedule P' is defined as

$$\bar{c}_{P'} = \bar{c}_{P^*} + c_{e(P^*)i} - \lambda_i \delta - \sum_k \Big(\mu_{k,i}(F_{k,|d_i|}(l_i^{P^*} + \delta) - F_{k,|d_i|}(l_i^{P^*}))+$$

$$\eta_{k,i}(F_{k,|d_i|}^{FS,1}(l_i^{P^*} + \delta) - F_{k,|d_i|}^{FS,1}(l_i^{P^*})) + \zeta_{k,i}(F_{k,|d_i|}^{CCM,1}(l_i^{P^*} + \delta) - F_{k,|d_i|}^{CCM,1}(l_i^{P^*})) \Big).$$

(3)

The computation of the lower bound $lb(\cdot)$ is adapted accordingly (it only modifies the cost of the transitions in the dynamic programming). In our implementation we only keep inequalities having $k = 2$ and $k = 3$.

4.2 Profitability cuts

Proposition 1 implies that a traversal between two pickup vertices or two delivery vertices is done at most once in each profitable schedule. Therefore, we considered the following clique inequalities:

$$\sum_{s \in \mathcal{S}_{i,j}^1} z^s \quad \leq \quad 1 \qquad \forall i, j \text{ vertices of same type}$$

where $\mathcal{S}_{i,j}^1$ is the set of all feasible and profitable schedules containing either the arc (i, j) or the arc (j, i).

Similarly, proposition 1 allows us to include the following inequalities:

$$\sum_{s \in \mathcal{S}_{i,j}^2} z_r \quad \leq \quad 1 \qquad \forall i, j \text{ vertices of distinct type}$$

where $\mathcal{S}_{i,j}^2$ is the set of all feasible and profitable schedules containing the arc (i, j) with a load strictly smaller than Q or strictly larger than 0 according to the type of vertices i and j.

We include the contribution of the dual variable of each of these inequalities in GENPATH in a similar way as in Section 4.1. We however neglect their contribution during the computation of $lb(\cdot)$, in order to keep the completion bounding procedure as fast as possible. Note that this might lead to generate more partial schedules in GENPATH as needed, but the optimal value of the pricing problem is not affected (and thus the value obtained for the relaxation is not affected neither).

5 Upper bounds

To obtain good feasible integer solutions quickly we designed two heuristics.

5.1 Iterative rounding heuristic

The first one is a simple iterative rounding scheme to be run when column generation is over. We consider the corresponding fractional solution, and fix to one a variable of highest value in that solution. Then, column generation is resumed, and the rounding process is iterated, until a feasible integer solution is found, M variables are fixed to one, or an infeasible LP is detected. We note that, when one of the latter two stopping conditions are met, such an iterative rounding fails in finding feasible solutions.

5.2 Memetic algorithm

The second heuristic is a memetic algorithm. Efficient encodings of the schedules are possible thank to the following proposition, generalizing Proposition 1 of [9] to the multiple vehicle case.

Proposition 3. *Consider an MVBP instance and suppose given M routes. Deciding whether there is a sequence of pickup and delivery operations for each route making the M routes a feasible solution of the MVBP can be performed in polynomial time. Moreover in case the answer is 'no', the minimum number of inventory units to remove from the supply and from the demand so that the answer becomes 'yes' can also be computed in polynomial time.*

In other words, when the answer is 'no', we can compute a new (d_i') in polynomial time satisfying (1) such that the answer becomes 'yes' while minimizing the number of inventory units removed.

Proof of Proposition 3. Let $R_1, \ldots, R_M$ be M routes. We indicate the vertex in position k of route m by r_k^m. For each $m = 1, \ldots, M$, let $\hat{G}^m = (\hat{V}^m, \hat{A}^m)$ be a digraph having one vertex for each occurrence of a vertex in R_m, and two types of arcs:

1. *route arcs*: one for each arc in R_m, connecting the corresponding vertices in $\hat{V}^m$

2. *load arcs*: one for each occurrence of a vertex in R_m, connecting the corresponding vertex in $\hat{V}^m$ to its next occurrence, if any.

We assign capacity Q to each route arc, capacity $+\infty$ to each load arc of pickup vertices, and capacity $-\infty$ to each load arc of delivery vertices. Each graph $\hat{G}^m$ resembles that used in the case of one vehicle in [9].

Now we define a support digraph $\hat{G} = (\hat{V}, \hat{A})$ as the disjoint union of all $\hat{G}^m$, and we add to $\hat{A}$ *inter-route load arcs*. These are defined as follows: for each $m = 1, \ldots, M$, and for the vertex i^m encoding the last occurrence of any vertex i in R_m, connect i^m to the vertex $i^{m'}$ encoding the first occurrence of i in a route $r^{m'}$, if any, where m' is chosen as small as possible with $m' > m$. We assign capacity $+\infty$ to all inter-route load arcs. Finally, we add to $\hat{V}$ a source vertex s and a sink vertex t; for each pickup vertex $i \in V$, we connect s to the first occurrence of i in $\hat{V}$ with a *initial arc* of capacity d_i, and for each delivery
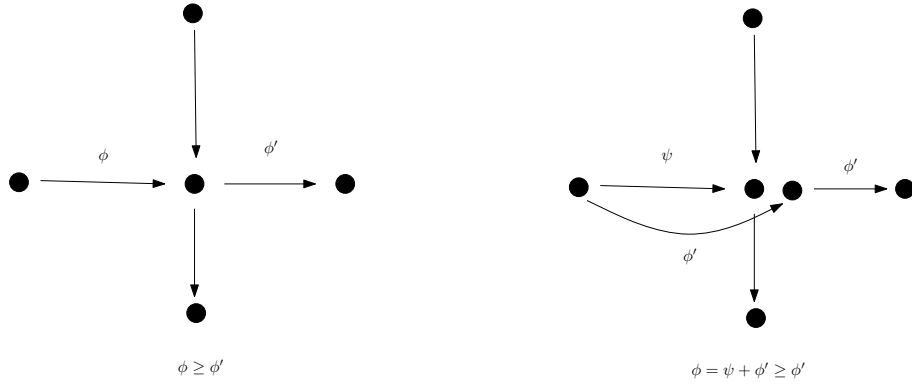
Figure 3: How to impose an inequality between the entering and the leaving flows in a vertex

vertex $i \in V$, we connect the last occurrence of i in $\hat{V}$ to t with a *target arc* of capacity $-d_i$.

Further, in order to ensure that inventory units are not loaded at delivery vertices, we replace such vertices as depicted in Figure 3. Such modification avoids subsequent route arcs with increasing flow on delivery vertices. It is also performed on pickup vertices to ensure that flow on their subsequent route arcs does not decrease. Now, there is a bijection between s-t flows on $\hat{G}$ and feasible rebalancing patterns.

The number of inventory units to be moved by vehicle m while going from r_k^m to r_{k+1}^m corresponds to the flow on route arcs; the number of inventory units remaining in a vertex i after each visit of the vehicle m corresponds to the flow on the arc between the corresponding occurrence and the next one, that is flow on either load or inter-route load arcs. The initial and target number of inventory units in each node corresponds to flow on starting and target arcs, respectively. Further, the modifications of Figure 3 forbid unbalancing.

In particular, a maximum flow in $\hat{G}$ corresponds to a pattern rebalancing the highest number of inventory units, as in the best case all starting and target arcs are saturated. Since maximum flows problems can be solved in polynomial time, so is the problem in the claim. □

5.2.1 The algorithm

Memetic algorithms, denoted MA in the remainder of the paper, are improved version of genetic algorithms and have been introduced by Moscato [18]. As for genetic algorithms, a population of Γ tentative solutions (individuals) is received in input and improved; a score is associated to each of them. The population is subject to two transformations: only best individuals are retained (selection)

and combined to create new individuals (cross-over), and local improvement is applied on each new individual to reach feasibility and improve its score.

Individuals   Proposition 3 enables to encode an individual as a simple list of M routes, one for each vehicle, each being in turn a list of integers representing the sequence of visited vertices. The score of an individual is set as its cost plus a penalty proportional to the total number of inventory units to remove in order to make it feasible, which is again computed as described in the proof of Proposition 3.

Selection and cross-over   At each main iteration a single individual is selected at random with a probability that is proportional to its score. After preliminary experiments we designed the following procedure. We split the population into two classes: the first containing 1/6 of the population, corresponding to the best individuals, and the second containing the remaining ones. Then, we perform a stratified sampling, selecting the first (resp. second) class with probability 70% (resp. 30%), and then picking an individual in the selected class with uniform probability. A second individual, instead, is chosen uniformly at random among those of the whole population, avoiding repetitions.

To perform cross-over, we adapt the route-first, cluster-second method of Prins [19]. To that purpose, a giant route is defined for each individual by appending its routes in sequence one after the other. Then, a cut point is selected, as a random integer from a uniform distribution between 1 and the length of the shortest of the two giant routes. Then, the first part of the giant route of the first individual is completed with the second part of the giant route of the second individual. All vertices missing from both parts are collected and organized in subsequence, respecting their precedences in the giant route of the second individual. Such a sequence of residual vertices is then inserted in between the first and the second part of the giant route of the new individual. The second part of the giant route of the first individual, and the first part of that of the second individual are combined similarly.

We get thus two giant routes. Each of them is then split into M routes, in order to get two new individuals. After local improvements performed on these new individuals, they are inserted in the population. To keep the population size constant, the two individuals with worst score are then removed. How a giant route is split into M routes to get a new individual and how the local improvements are performed is explained in the next paragraph.

Giant route splitting and local improvement   The one-vehicle version of Proposition 3 [9] is used as a preliminary feasibility test on the giant route. If no feasible rebalancing pattern can be obtained, then no feasible MVBP solution can be obtained after splitting: a score $+\infty$ is assigned to the new individual.

When the preliminary test is successful, the new giant route is split using an adaptation of the procedure of the aforementioned paper by Prins. We define an acyclic directed graph $\widetilde{G}$ whose vertices are that of the giant route,

therefore possibly including multiple occurrences of vertices of the original set V. An arc (i, j) is created between two vertices if (a) there are at most $(T - 3)$ vertices between them on the giant route (b) i is a pickup vertex, j is a delivery vertex, and the depot is not between i and j in the giant route (c) there are no consecutive occurrences of the same vertex between i and j in the giant route. Condition (a) ensures that the sequence between the two endpoints of such an arc can lead to a route respecting the maximum length constraint, while (b) and (c) avoid unprofitable schedules since a solution is always suboptimal if it is composed of routes visiting the depot or having self-loops.

We set as weight of each arc (i, j) of $\tilde{G}$ the cost of the route starting and ending with the depot and going through the sequence of vertices between i and j on the giant route. The split is then obtained by computing a directed path in $\tilde{G}$ of minimum weight, using at most M arcs, that can efficiently be calculated by using dynamic programming.

Finally, local search is applied to the set of routes to reach a local minimum. We included three types of moves: intra-route 2-OPT, inter-route 2-OPT and CLEANING. In particular, we iteratively explore both intra-route 2-OPT and inter-route 2-OPT neighbourhoods with a best-improvement policy, and we apply only the best move among the two neighbourhoods. When no improving move is found we proceed to CLEANING, that is we modify the solution in order to fulfill Feature 1s, 2s, and 3s.

While the giant route splitting is performed by considering route costs only, each local improvement move is evaluated by computing also the best associated rebalancing pattern, using the flow algorithm described in Proposition 3, and thus obtaining the actual score of the individual.

Multistart and termination After a fixed number of selection and crossover iterations we re-initialize the population as follows. First, we drop a fraction α of the population, that is we keep only the best $\alpha \cdot \Gamma$ individuals. Then, we randomly generate $\beta \cdot \Gamma$ new individuals and perform local improvement on them. Finally, we keep only the overall best Γ individuals. The MA is stopped after a fixed number of restarts.

Implementation After preliminary experiments we set the initial population size to $\Gamma = 30$, generating the individuals differently in different calls of our MA, as described in Section 6. We restart every 100 main iterations, and we stop the MA after 6 restarts. At each restart we keep only a fraction $\alpha = 0.5$ of the population, and generate a fraction $\beta = 0.8$ of new individuals.

6 Algorithms

We embed our lower and upper bounds into the following optimization framework.

(a) Initialize the MA of Section 5 with random routes, and run it, obtaining an initial upper bound **M1** and a population of good schedules.

(b) Solve (RP) with the column generation procedure described in Section 3 initialized with the schedules obtained in Step (a). Add cuts as described in Section 4, and reoptimize with column generation. At the end, we have a lower bound **LB**.

(c) Perform the iterative rounding heuristic. If a feasible MVBP solution is obtained, whose value matches **LB**, then stop: optimality is proved.

(d) Calculate better feasible MVBP solutions with the MA of Section 5, starting from a population including the final population of Step (a), and the solution of Step (c) if any, obtaining an upper bound **M2**. If **M2** = **LB** then stop: optimality is proved.

(e) Generate all schedules whose reduced cost does not exceed $\rho = \mathbf{M2} - \mathbf{LB}$, using GENPATH and GENROUTE (described in Section 3).

(f) Restrict $\mathcal{S}$ to schedules generated in Step (e) and solve to optimality the resulting restricted integer program with the help of a general purpose solver.

From a theoretical point of view, our algorithm yields global optimal solutions (see Baldacci *et al.* [7]); however, our experiments show that it unfolds its potential as a matheuristic, by making the following modifications: in Step (b), we always stop GENPATH when the limit of 15'000 paths has been reached (i.e. we stop after the first round of Section 3.4); we moreover skip Step (e) and give to the solver the whole set of schedules generated in Step (b) and all schedules of the best individual obtained at the end of Step (d). When 15'000 is reached without having found a negative reduced cost schedule in Step (b), we just take the fractional solution of the last LP as the initial fractional solution for the iterative rounding heuristic.

7 Computational study and conclusions

All the algorithms have been coded in C++ and tested on a PC equipped with an Intel Core i7 3770 CPU clocked at 3.40GHz and 8 GB of RAM. CPLEX 12.4 has been used for both solving the LP subproblems and as a general purpose IP solver. All CPLEX options have been kept at default values, except for multi-threading support, which has been turned off.

Our benchmark consists of the instances of [9] with up to 30 vertices. These are randomly generated networks with vertices located in a two-dimensional square of coordinates in $[-500, 500] \times [-500, 500]$, the depot being located at $(0, 0)$. Travelling costs c_{ij} are computed as the Euclidean distances between vertices i and j. The demands of $n - 1$ vertices are randomly drawn from a uniform distribution as an integer demand between $[-10, 10]$, where positive

values define pickup vertices, and negative values define delivery ones, while the demand of the nth vertex is set to $-\sum_{i=0}^{n-1} d_i$.

Additionally, we include in our benchmark new smaller instances with 10 vertices that are obtained by splitting instances with 20 vertices in half, with respect to the order given in the input file. The full dataset consists therefore of 40 instances. The vehicle capacity Q was set to 10, the number $|M|$ of available vehicles to 5, and the the maximum number T of stops in each route to 10.

To improve the numerical stability during column generation, in (RP) we relax constraints (i) in $\geq$ form, thereby restricting λ_i dual variables to be nonnegative. From a theoretical point of view, this comes at the expense of a possible weakening in the quality of the relaxation. However, preliminary experiments showed that such a weakening is negligible in all instances but one, while stability is substantially improved. In Step (f) however, constraints (i) are considered as in the original form, with an equality. In the appendix we report a detailed discussion on this issue.

7.1 Solving instances to proven optimality

In a first run of experiments we test our approach as an exact optimization algorithm. In Table 1 we report the results obtained with our implementation, setting a time limit of 3 hours for each run. For each instance we report the number **n** of vertices with demand $d_i \neq 0$, the upper bound **M1** found after the first call of the memetic algorithm (Step (a)), the lower bound **LB** found by the column generation procedure (Step (b)), the upper bound **RO** found after the iterative rounding heuristic (Step (c)) and the upper bound **M2** found after the second call to the memetic algorithm (Step (d)), the final upper bound **UB** computed by solving the restricted IP with the solver (Step (f)), the gap **g** between **UB** and **LB**, and the overall computing time (**t**). The instances solved to proven optimality by the solver (in Step (f) in Section 6) are marked with a star symbol in the **UB** column, and those whose computation hit the time limit are marked with a dash in the time column. Similarly, a dash symbol indicates those bounds that could not be computed within the time limit.

We observe that our algorithm solves to proven optimality all the instances with 10 vertices and 4 instances with 20 vertices. The iterative rounding heuristic always provides a feasible solution when it was possible to compute the **LB**. Moreover, the value found by this heuristics is in almost all cases the best one found by the overall method. The memetic algorithms is effective, finding an optimal solution at the first call on around 37% of the instances. Such value grows to 45% if we also consider the second call. Furthermore, the lower bound given by the column generation procedure is of excellent quality for instances up to 20 vertices, and the average gap for the instances for which this lower bound has been computable is less than 10%.

Instance	n	M1	LB	RO	M2	UB	g (%)	t (s)
n10q10A	10	3055	3055.00	3055	3055	3055*	-	38.52
n10q10a	10	3719	3611.25	3719	3719	3719*	2.98	93.38
n10q10B	10	3745	3631.95	3745	3745	3704*	1.98	98.01
n10q10b	10	3192	3114.50	3192	3192	3192*	2.49	66.31
n10q10C	10	3392	3258.00	3392	3392	3392*	4.11	5759.59
n10q10c	10	4239	4239.00	4239	4239	4239*	-	37.27
n10q10D	10	3273	3147.33	3273	3273	3199*	1.64	486.33
n10q10d	10	4497	4206.83	4497	4497	4497*	6.90	107.51
n10q10E	10	4921	4857.29	4921	4921	4876*	0.39	252.86
n10q10e	10	3846	3675.60	3835	3823	3823*	4.01	264.34
n10q10F	10	4044	3794.88	4044	3796	3796*	0.03	44.13
n10q10f	10	3468	3283.46	3468	3468	3468*	5.62	73.33
n10q10G	10	4151	3735.61	4151	4151	3973*	6.35	229.8
n10q10g	10	4179	4075.20	4179	4179	4179*	2.55	81.51
n10q10H	10	3959	3772.06	3959	3959	3959*	4.96	715.83
n10q10h	10	4168	4043.89	4168	4168	4168*	3.07	2211.82
n10q10I	10	4026	3711.20	4026	4026	3963*	6.78	119.52
n10q10i	10	2677	2632.50	2677	2677	2645*	0.47	2867.46
n10q10J	9	3125	3125.00	3125	3125	3125*	-	285.28
n10q10j	10	3453	3392.40	3453	3453	3453*	1.79	96.8
n20q10A	17	4826	4758.00	4826	4826	4826*	1.43	1102.45
n20q10B	18	5391	5051.63	5391	5391	5391	6.72	-
n20q10C	20	6508	6317.00	6508	6508	6508	3.02	-
n20q10D	19	6399	6208.00	6208	6208	6208*	-	509.07
n20q10E	18	6632	6216.77	6632	6506	6506	4.65	-
n20q10F	20	5222	5162.82	5222	5222	5222*	1.15	863.68
n20q10G	19	5867	5466.36	5867	5867	5867	7.33	-
n20q10H	18	6546	5824.36	6546	-	6546	12.39	-
n20q10I	19	5279	4879.63	5279	5279	5279	8.18	-
n20q10J	16	4545	4545.00	4545	4545	4545*	-	770.56
n30q10A	29	7178	6637.53	7178	-	7178	8.14	-
n30q10B	25	7274	-	-	-	7274	100.00	-
n30q10C	28	7513	-	-	-	7513	100.00	-
n30q10D	28	8024	6729.90	8024	-	8024	19.23	-
n30q10E	26	7136	6497.56	7136	-	7136	9.83	-
n30q10F	29	7014	6450.80	7014	-	7014	8.73	-
n30q10G	27	9819	8879.76	9819	-	9819	10.58	-
n30q10H	27	7705	6640.78	7705	7458	7458	12.31	-
n30q10I	27	6462	-	-	-	6462	100.00	-
n30q10J	27	7092	-	-	-	7092	100.00	-

Table 1: Results on the instances with up to 30 stations.

7.2 Using the algorithm as a matheuristic

In a second run we test our approach when turned into a matheuristic as described in the Section 6. In Table 2 we report for each instance the number of vertices with demand $d_i \neq 0$ (**n**), the upper bound **UB** obtained previously, the upper bound **UB'** obtained by the matheuristic, the gap between them (**g**), and the computing time (**t**) in seconds.

For what concerns solutions quality, we first observe that our algorithm finds the optimal solution for almost 50% of the instances. Furthermore, for more than 70% of the instances UB' is equal to UB, while in the remaining ones the gap is always less than 5%. We also observe that for instances n20q10I, n30q10A, n30q10G, and n30q10J our algorithm used as matheuristic improves the UB obtained as an exact approach. Such results are obtained within an average computing time that is an order of magnitude smaller than the exact approach one.

7.3 Solving the Split Delivery VRP

The MVBP is a generalization of the SDVRP as noted in Section 1 and in principle our algorithm may be used to solve it by a simple remapping of the instances. Of course, many special purpose techniques, tailored to the SDVRP, cannot be applied to the MVBP. Therefore, in an effort of evaluating the computational price of the additional modeling flexibility of MVBP, in a third run of experiments we consider the set of instances of the SDVRP taken from [2] with at most 50 vertices. In a simple preprocessing step we rescale both capacities and demand coefficients by a factor 10, obtaining capacity 10 for each vehicle; then, the bound T is also set to 10, since at least one unit of capacity is spent whenever a customer is visited. Also, we consider an unbounded number of vehicles $|M|$, and therefore constraint (ii) becomes useless in formulations (P) and (RP). To match the MVBP instance structure, a pickup vertex at a zero-distance from the depot is also added, containing all the commodities to be delivered at the other vertices.

In Table 3 we report the results obtained by our exact approach with a maximum computing time of 6 hours. For each instance we report the number of vertices with demand $d_i \neq 0$ (**n**), the final lower and upper bounds given by our algorithm (**LB** and **UB**, respectively), the gap between **LB** and **UB** (**g**), the computing time (**t**), the optimal solution of the instance as reported in [2] (**z***), and the gaps from **LB** and **UB** to **z*** (**g** $\frac{\mathbf{LB}}{\mathbf{z*}}$ and **g** $\frac{\mathbf{UB}}{\mathbf{z*}}$, respectively).

Our algorithm is able to solve to optimality three instances out of nine. Furthermore, we observe that the average gap between **LB** and **z*** is small, proving the effectiveness of our bounding procedure also for the SDVRP.

Instance	n	UB	UB'	g(%)	t(s)
n10q10A	10	3055	3055	0.00	52.73
n10q10a	10	3719	3719	0.00	52.65
n10q10B	10	3704	3745	1.11	44.53
n10q10b	10	3192	3192	0.00	56.60
n10q10C	10	3392	3392	0.00	149.65
n10q10c	10	4239	4239	0.00	38.60
n10q10D	10	3199	3273	2.31	76.15
n10q10d	10	4497	4497	0.00	39.89
n10q10E	10	4876	4921	0.92	51.14
n10q10e	10	3823	3823	0.00	130.33
n10q10F	10	3796	3796	0.00	87.35
n10q10f	10	3468	3468	0.00	38.32
n10q10G	10	3973	3973	0.00	74.74
n10q10g	10	4179	4179	0.00	47.49
n10q10H	10	3959	3959	0.00	76.12
n10q10h	10	4168	4168	0.00	73.10
n10q10I	10	3963	4026	1.59	50.12
n10q10i	10	2645	2645	0.00	117.96
n10q10J	9	3125	3125	0.00	72.62
n10q10j	10	3453	3453	0.00	56.15
n20q10A	17	4826	4826	0.00	326.66
n20q10B	18	5391	5391	0.00	442.33
n20q10C	20	6508	6508	0.00	488.61
n20q10D	19	6208	6399	3.08	469.08
n20q10E	18	6506	6632	1.94	295.27
n20q10F	20	5222	5222	0.00	828.72
n20q10G	19	5867	5867	0.00	499.47
n20q10H	18	6546	6546	0.00	440.93
n20q10I	19	5279	5227	-0.99	481.45
n20q10J	16	4545	4545	0.00	318.77
n30q10A	29	7178	7065	-1.57	1819.89
n30q10B	25	7274	7274	0.00	1302.04
n30q10C	28	7513	7513	0.00	1329.45
n30q10D	28	8024	8024	0.00	1196.11
n30q10E	26	7136	7136	0.00	1228.42
n30q10F	29	7014	7014	0.00	1744.97
n30q10G	27	9819	9766	-0.54	1914.57
n30q10H	27	7458	7705	3.31	966.85
n30q10I	27	6462	6462	0.00	1490.14
n30q10J	27	7092	7081	-0.16	2165.14

Table 2: Results on the instances with up to 30 stations.

Instance	n	LB	UB	g (%)	t (s)	z*	g $\frac{\text{LB}}{\text{z*}}$(%)	g $\frac{\text{UB}}{\text{z*}}$(%)
SD1	9	21555.56	22828*	5.90	1878.71	22828	5.90	0.00
SD2	17	66475.99	72000	8.31	-	70828	6.55	1.65
SD3	17	42840.00	43040*	0.47	2592.71	43040	0.47	0.00
SD4	25	62895.56	63062*	0.26	18450.39	63062	0.26	0.00
SD5	33	132514.59	149935	13.15	-	138994	4.89	7.87
SD6	33	82988.89	95527	15.11	-	83086	0.12	14.97
SD7	41	345261.00	402000	16.43	-	364000	5.43	10.44
SD8	49	486218.50	550000	13.12	-	506828	4.24	8.52
SD9	49	197319.03	214962	8.94	-	204288	3.53	5.22

Table 3: Results on Split Delivery VRP instances with up to 50 stations.

8   Conclusions

In this paper we introduced a new routing problem, the Multiple Vehicle Balancing Problem, which generalizes already difficult routing problems such as the Split Delivery Vehicle Routing Problem. Using theoretical properties of the problem, we propose an integer program model and an optimization framework that can be used both as an exact algorithm and as a matheuristic. When used as an exact approach, our algorithm could solve to proven optimality instances up to 20 vertices; when used as a matheuristic, it unfolded its potential, solving instances with up to 30 vertices in reasonable computing time. Our methods prove to be a viable tool also for solving instances of the simpler Split Delivery Vehicle Routing Problem, providing an appealing trade-off between modeling flexibility and numerical solution effectiveness.

Even if our contributions are mainly methodological, we can also discuss their practical interest. Instances of real systems of bicycle sharing may include thousands of stations, but those actually requiring optimized rebalancing operations may be a substantially restricted subset. When such conditions holds, our methodology might be used to optimize even real case instances, being able to automatically discard irrelevant vertices in a preprocessing phase.

References

[1] C. Archetti, N. Bianchessi, and M.G. Speranza. Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238:685–698, 2014.

[2] C. Archetti, N. Bianchessi, M.G. Speranza, and A. Hertz. A column generation approach for the split delivery vehicle routing problem. *Networks*, 58:241–254, 2011.

[3] C. Archetti and M.G. Speranza. An overview on the split delivery vehicle routing problem. In K.-H. Waldmann and U.M. Stocker, editors, *Oper-*

ations Research Proceedings 2006, pages 123–127. Springer Berlin Heidelberg, 2007.

[4] C. Archetti, M.G. Speranza, and A. Hertz. A Tabu Search Algorithm for the Split Delivery Vehicle Routing Problem. *Transportation Science*, 40:64–73, 2006.

[5] R. Baldacci, E. Bartolini, and A. Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59:414–426, 2011.

[6] R. Baldacci, M. Battarra, and D. Vigo. Routing a heterogeneous fleet of vehicles. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 3–27. Springer, 2008.

[7] R. Baldacci, N. Christophides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

[8] J. Carlier, F. Clautiaux, and A. Moukrim. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers and Operations Research*, 34:2223–2250, 2007.

[9] D. Chemla, F. Meunier, and R. Wolfler Calvo. Bike-sharing system: solving the static rebalancing problem. *Discrete Optimization*, 10:120–146, 2013.

[10] F. Clautiaux, C. Alves, and J.M. Valério de Carvalho. A survey of dual-feasible and superadditive functions. *Annals of Operations Research*, 179:317–342, 2010.

[11] M. Dell'Amico, E. Hadjicostantinou, M. Iori, and S. Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19, 2014.

[12] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Springer, 2005.

[13] M. Dror and P. Trudeau. Split delivery routing. *Naval Research Logistics*, 37:383–402, 1990.

[14] F. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31, 2001.

[15] I.A. Forma, T. Raviv, and M. Tzur. A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research, Part B*, 71:230–247, 2015.

[16] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In *Column Generation*, pages 33–65. Springer, 2005.

[17] G. Laporte, F. Meunier, and R. Wolfler Calvo. Shared mobility systems. *4OR*, 13:341–360, 2015.

[18] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms. Technical report, 1989.

[19] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, pages 1985 – 2002, 2004.

[20] T. Raviv, M. Tzur, and I.A. Forma. Static repositioning in a bike-sharing system: Models and solution approaches. *EURO Journal on Transportation and Logistics*, 2:187–229, 2013.

[21] V. Schmida, K.F. Doerner, R.F. Hartl, and J.-J. Salazar-González. Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. *Computers and Operations Research*, 37:559–574, 2009.

[22] J. Schuijbroek, R.C. Hampshire, and W.-J. van Hoeve. Inventory rebalancing and vehicle routing in bike sharing systems. Technical report, Tepper School of Business, 2013.
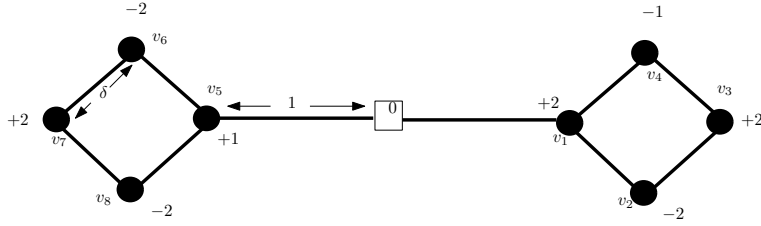
Figure 4: With $Q := 2$, $M := +\infty$, $T := 5$, an example of instance for which the optimal values of the relaxations $(\mathrm{RP})^=$ and $(\mathrm{RP})^{\geq}$ differ

Appendix

This appendix deals with the impact of relaxing the equality to inequality constraints in the relaxation proposed in Section 3. We denote as $(\mathrm{RP})^=$ and $(\mathrm{RP})^{\geq}$ the two linear programs. While in traditional Vehicle Routing Problems, this change has no impact on the optimal value of the relaxation, in our MVBP it could.

In fact, consider a network as that of Figure 4. The number near each vertex i corresponds to d_i. The number near each edge, instead, represents the shortest-path distance between the two endpoints, that is 1 for the edges incident to the depot, and $\delta \ll 1$ for the others. The other parameters are set as follows: $Q := 2$, $M := +\infty$, $T := 5$.

It is easy to check that no schedules except the following may be selected in an optimal solution of either $(\mathrm{RP})^=$ or $(\mathrm{RP})^{\geq}$.

	r^s	l^s	c^s
s_1	(v_1, v_2, v_3, v_4)	$(2, 2, 1, 1)$	$2 + 4\delta$
s_2	(v_1, v_4, v_3, v_2)	$(1, 1, 2, 2)$	$2 + 4\delta$
s_3	(v_5, v_6, v_7, v_8)	$(1, 1, 2, 2)$	$2 + 4\delta$
s_4	(v_5, v_8, v_7, v_6)	$(1, 1, 2, 2)$	$2 + 4\delta$
s_5	(v_1, v_2, v_3, v_6)	$(2, 2, 2, 2)$	$4 + 6\delta$
s_6	(v_1, v_4, v_3, v_6)	$(2, 1, 1, 2)$	$4 + 6\delta$

For $(\mathrm{RP})^{\geq}$, the primal solution $z^{s_1} = 1/2$, $z^{s_2} = 1/2$, $z^{s_3} = 5/6$, $z^{s_4} = 1/3$, $z^{s_5} = 1/4$ and $z^s = 0$ elsewhere is feasible and has a total cost of $16/3 + 61/6\delta$. The bottom network of Figure 5 shows the dual values associated to each vertices for in the corresponding dual solution. It is easy to check that they satisfy the constraints of the dual of $(\mathrm{RP})^{\geq}$, proving optimality.

Such a solution is infeasible for $(\mathrm{RP})^=$; instead, the primal solution $z^{s_1} = 1/4$, $z^{s_2} = 1/2$, $z^{s_3} = 1$, $z^{s_5} = 1/4$, $z^{s_6} = 1/4$ and $z^s = 0$ elsewhere, whose corresponding dual solution is depicted in Figure 5 (top), is optimal and has a total cost of $11/2 + 10\delta$, that when $\delta < 1$ is greater than $16/3 + 61/6\delta$.
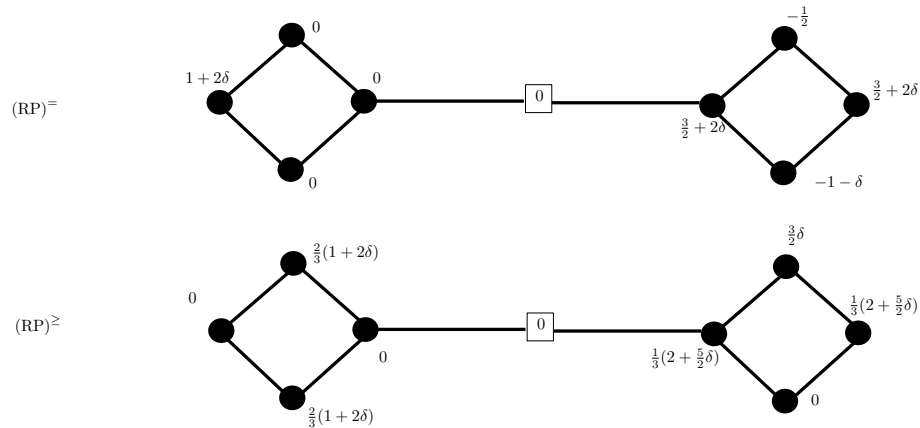
(RP)$^=$

(RP)$^\geq$

Figure 5: Feasible dual solutions of $(RP)^=$ and $(RP)^\geq$, which can be proved to be optimal

We finish with a proof that no optimal schedule for (P) can be missing from the set generated in Step (e) of the full optimization algorithm, even if the reduced costs are computed according to an optimal $(RP)^\geq$ solution instead of an optimal $(RP)^=$ one, as we do in our implementation. To that purpose, consider $(P)^\geq$ as the version of (P) in which constraints (i) are considered with $\geq$. It is the integer version of $(RP)^\geq$. Let s be a schedule present in an optimal solution of (P). This optimal solution of (P) has a cost non greater than **M2** and is a feasible solution of $(P)^\geq$. The reduced cost of s with respect to an optimal basis of $(RP)^\geq$ is thus at most $\rho = \mathbf{M2} - \mathbf{LB}$ and will thus be generated in Step (e). The proof is written here without speaking about the added valid inequalities, but the argument remains exactly the same.