A route decomposition approach for the single commodity Split Pickup and Split Delivery Vehicle Routing Problem

Marco Casazza[a,*], Alberto Ceselli[a], Roberto Wolfler Calvo[b]

[a] *Dipartimento di Informatica, Università degli Studi di Milano, Italy*
[b] *Laboratoire d'Informatique Paris Nord, Université Paris 13, France*

Abstract

We address a single commodity Pickup and Delivery Vehicle Routing Problem from the literature. A network of customer nodes is given with both travel times and costs. A fleet of vehicles of limited capacity is exploited to satisfy node demands, and a set of routes must be found such that the vehicle capacities are never exceeded, each route does not exceed time resource, and cost is minimized. The demands of both pickup and delivery nodes can be split, and each node can be visited more than once. We provide new theoretical insights. We propose a new formulation where routes are decomposed into sequences of simpler substructures called *clusters*, mitigating the combinatorial explosion of feasible solutions. We introduce valid inequalities, and design a branch-and-price algorithm, exploiting ad hoc pricing routines and branching strategies, and embedding a rounding heuristic to speed up pruning. An extensive experimental analysis shows our method to offer simultaneously more modelling flexibility and more computational effectiveness than previous attempts from the literature. Our investigation opens also interesting insights into the use of route decomposition strategies.

Keywords: routing, split pickup split delivery, branch-and-price, route decomposition, bike sharing

1. Introduction

In this paper we address a *single commodity Split Pickup and Split Delivery Vehicle Routing Problem* (SPSDVRP), a variant of the *Pickup and Delivery VRP* where node demands can be split.

We suppose that we are given a network where customers are placed, and where travelling implies costs and requires time. For each customer we are given

*Corresponding author
Email addresses: marco.casazza@unimi.it (Marco Casazza), alberto.ceselli@unimi.it (Alberto Ceselli), wolfler@lipn.fr (Roberto Wolfler Calvo)

a demand, that is the amount of commodity that he/she asks to be either picked up or delivered.

A fleet of identical vehicles, having limited capacity, is available to visit them. The demand of each customer must be fully satisfied, potentially visiting him/her more than once and with more than a single vehicle, if needed or fruitful. For each vehicle we are therefore required to find a route, that is a pattern defining which customers are visited, the order of their visits, and the amount of commodity loaded or unloaded at each of them. Each vehicle starts and ends empty at a depot, its capacity can never be exceeded. The overall travel time of a route must not exceed a time limit, that is meant to model a work shift. Finally, we assume that no customer is used as a temporary storage location, and therefore the amount of commodity in each station is always monotone.

The SPSDVRP requires to find a set of routes of minimum cost respecting the above conditions.

From an application point of view the SPSDVRP arises in bike sharing contexts where, during peak hours, flows along particular direction are registered, leading to both empty racks in departure stations, and full racks at destination. Both represent a disservice. One of the solutions chosen by many operators is to rebalance the system by moving bikes between stations using a fleet of trucks. The interested reader can find in Laporte et al. (2015) an updated survey on open problems in bike sharing contexts and in Espegren et al. (2016) a survey on rebalancing problems.

From a methodological point of view, the SPSDVRP is NP-hard and it belongs to the wide class of Pickup and Delivery VRPs (PDVRP). We refer to Battarra et al. (2014) for a recent survey about PDVRPs on transportation of freights. We also mention Gschwind et al. (2018) and Baldacci et al. (April 2011), that well represent the state of the art of exact algorithms. Both consider PDVRPs with Time Windows (PDPTW), and are based on column generation approaches.

Furthermore, in Pessoa et al. (2019) a generic branch-and-cut-and-price framework is introducted, that solves a formulation encoding a wide class of VRPs, including the PDPTW. A comparison against state-of-the-art ad-hoc approaches is also carried out.

Our SPSDVRP generalizes at the same time well known transportation models like the Split Delivery VRP. In the last years there has been a lively research interest in that field. Although showing a combinatorial nature similar to problems from the literature, the SPSDVRP stands out for a few features, making its solution particularly involved.

The first work addressing the SPSDVRP was Raviv et al. (2013), where the authors propose several models to maximize user satisfaction and a two-phase heuristic. Their algorithm was then outperformed by a three-step matheuristic in Forma et al. (2015). Still regarding heuristic methodologies, in Rainer-Harbach et al. (2013) the authors consider a multiobjective function minimizing the amount of unsatisfied demands, the number of loading and unloading operations, and the routing cost. They propose a Variable Neighbourhood Search approach exploring various neighbourhoods, and they later improve their work

in Rainer-Harbach et al. (2015). The same model is exploited in Di Gaspero et al. (2013) were the authors propose a hybrid metaheuristic that combines Ant Colony Optimization and Constraint Programming into a bi-level optimization algorithm. In Alvarez-Valdes et al. (2016) a two-phased rebalancing heuristic algorithm is proposed that first identifies which pickup station supplies which delivery station, and then constructs routes in a greedy fashion.

For what concerns the exact methodologies for SPSDVRP, in Espegren et al. (2016) the authors propose a new formulations and valid inequalities when the satisfaction of nodes demands is not a constraint but a cost in the objective function, while in Bulhões et al. (2018) the SPSDVRP is addressed imposing that each node must be visited by the same vehicle only and considering an additional service time for loading and unloading operations. In the latter, a branch-and-cut approach is proposed that solves to optimality instances with up to 41 nodes.

Indeed in Casazza et al. (2018a) we addressed the SPSDVRP in its core features, proposing a mathematical programming approach where we modelled the problem by means of a set partitioning extended formulation in which each column represents a route including all the loading information. However, our previous methodology left two open issues. First, from a modelling point of view, it was not designed to explicitly handle travel times: these were instead approximated by a limit on the number of customer visits in each route. Second, from an algorithmic point of view, due to the nature of the pricing procedure, computing times were extremely sensitive to the magnitude of customer demand and vehicle capacity coefficients.

Unfortunately, we did not find means of overcoming these issues by simply extending our previous approach. In this paper we therefore propose a new mathematical programming model, and a corresponding branch-and-price-and-cut algorithm for the SPSDVRP that proves to overcome these two issues, and to provide better performances than Casazza et al. (2018a) at the same time. The main idea is to decompose routes into simpler substructures, thereby proving and exploiting a rich set of combinatorial properties on them. A preliminary version of our approach, covering partial results, was presented as Casazza et al. (2018c).

In Section 2 we first formalize the problem and then, in Subsection 2.1, we describe the way we decompose vehicle routes, obtaining simpler combinatorial substructures that we call *clusters*, investigating their theoretical properties. In Section 3 we propose a new extended formulation that combines these substructure to reduce the complexity of the problem along with valid inequalities to improve the quality of the dual bound provided by our formulation. In Section 4 we explain the details of our methodology, describing how we obtain dual bounds by means of column generation procedures, and how we integrate such procedures into a branch-and-price framework, along with ad-hoc branching strategies and primal heuristics. Finally, in Section 5 we study the performances of our algorithm by means of an extensive experimental campaign. In Section 6 we collect some conclusions.

2. Models and structural properties

The SPSDVRP can be formalized as follows: We are given a directed graph $G = (N_0, A)$, where $N_0 = \{0, \dots\}$ is a set including both the customer nodes $N = \{1, \dots\}$ and a depot 0, and $A = \{(i, j) \mid i, j \in N_0\}$ is a set of arcs connecting them. Let c_{ij} and t_{ij} be the travelling cost and time of arc $(i, j) \in A$, respectively. We also assume that both costs and times satisfy triangular inequality.

Customer nodes can be partitioned into two sets: the set of pickup nodes N^+ and the set of delivery nodes N^-. For each customer node $i \in N$, we are given a demand d_i, that is the amount of commodity to be picked up if $i \in N^+$, or to be delivered if $i \in N^-$. We assume that $\sum_{i \in N^+} d_i = \sum_{i \in N^-} d_i$.

A homogeneous fleet, represented by a set of vehicles of limited capacity $M = \{1, \dots\}$, is given. Vehicles are requested to carry units of commodity from pickup nodes to delivery ones in order to fulfil exactly their demands. Each node can be visited more than once by one or more distinct vehicles.

We define a *route* as a sequence of nodes visited by a vehicle that starts and ends at the depot. We say that arc (i, j) is traversed in the route for each pair of nodes i and j which are consecutively visited. The travelling cost and the time of a route are the sum of the costs and the sum of the times of the arcs traversed, respectively. A route is feasible only if it does not exceed a given time limit T.

We define the *loading plan* of a vehicle as the amount of demand loaded (resp. unloaded) at each pickup node (resp. delivery node) visited.

Finally, we define a *routing plan* as a route along with a loading plan. A routing plan is feasible only if its corresponding route is feasible and the amount of demand loaded on the vehicle after visiting a node in the route is always non-negative and never exceeds a capacity Q.

The SPSDVRP aims to find a minimum cost set of routing plans satisfying node demands.

In Figure 1 we depict an example in which we assume 2 vehicles with capacity $Q = 10$ each. Both vehicles start empty from the depot and visit pickup node 6 splitting its demand. The load of vehicles after each operation is reported as a label on the arcs of the solutions. Also pickup node 5 is visited twice, but this time by the same vehicle. The vehicles continue independently their routes until they reach the delivery node 11, where they fractionally serve its demand before ending their routes empty at the depot.

2.1. Routing plans and clusters

We first present some observations that highlight particular regularities of the routing plans. These are subsequently exploited to devise combinatorial properties and a new extended formulation.

Since the objective of our problem is to minimize the total travelling costs of the selected routing plans, we first observe that, due to triangular inequality:

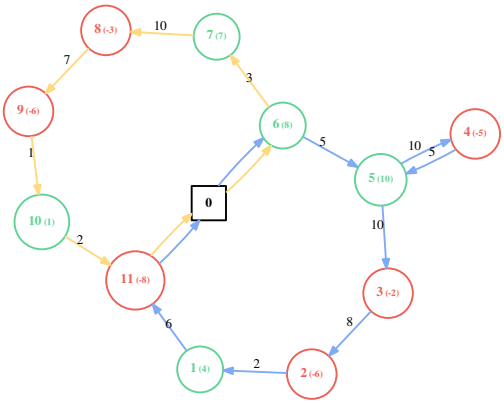

Figure 1: Example of feasible solution assuming a capacity $Q = 10$ and using 2 vehicles. Each arc has a label reporting the load of the vehicle after visiting a node. Nodes 5, 6, and 11 are visited twice splitting their demands.

Observation 2.1. *There always exists an optimal routing plan where no pickup node (resp. delivery node) is visited without collecting (resp. delivering) at least one unit of its demand.*

Furthermore, since vehicles always start and end empty at the depot and nodes cannot serve as temporary storages, we can observe that:

Observation 2.2. *There always exists an optimal routing plan where the first and the last nodes of each route are a pickup and a delivery node, respectively.*

Therefore, a route is an alternate sequence of pickup and delivery nodes, that always starts with a pickup, and always ends with a delivery.

Our intuition is therefore that the structure of a route can be much simplified by explicitly encoding such an interleaved behaviour. Indeed, in the following we formalize such an intuition, and prove a few key properties of such an encoding.

Definition 2.1. *We denote a* cluster *either as a single depot node or as a non-empty sequence of pickup nodes followed by a non-empty sequence delivery nodes along with their loaded (resp. unloaded) demands.*

We remark that each node in a cluster must be adjacent in G to both its preceding and its following node in the cluster, if any.

In other words, a cluster is a slice of a routing plan. Therefore, a route is itself a sequence of one or more clusters linked together, plus an additional stop at the depot at the beginning and at the end. An example of cluster structure in a route is depicted in Figure 2: the first route is partitioned in three clusters, two with three nodes and one with two nodes. The second route is partitioned in two clusters, one with four and one with two nodes. All clusters start with a pickup node and end with a delivery node.

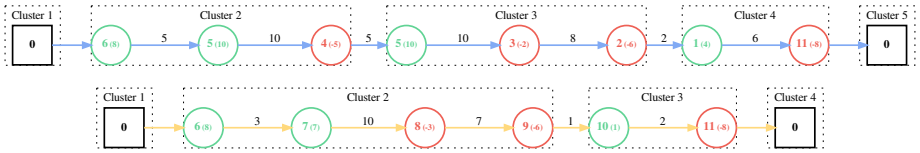

Figure 2: Cluster partitioning of routes in Figure 1

Definition 2.2. *Given a cluster in a route, such cluster is* feasible *if both the sum of loaded demands and the sum of unloaded demands do not exceed the capacity Q.*

Definition 2.3. *The* cost of a cluster *is the sum of the arcs connecting the nodes of the inner pickup and delivery sequences.*

Let c and c' be two consecutive clusters of a route, and let i and j be the last and first node of cluster c and c', respectively. Then we say that clusters c and c' are connected by a *linking* arc (i, j).

Definition 2.4. *The* cost of a route *is the sum of the cost of its clusters and the cost of its linking arcs.*

We readily observe that:

Proposition 2.1. *There always exists an optimal SPSDVRP solution where no node is visited more than once in the same cluster.*

Proof. In fact, let us suppose by contradiction that such an optimal solution exists and that i is a pickup node visited twice in a cluster. Let q'_i and q''_i be the two quantities loaded on the vehicle, and that $\tilde{q}_i$ is the demand of i loaded, that is $\tilde{q}_i = q'_i + q''_i$. Since the visits occur in the same cluster, we know that between the first and the second visit there are only pickup nodes, and that the capacity is not exceeded. Therefore, we can set $q'_i = \tilde{q}_i$ and $q''_i = 0$, avoiding the second visit due to Observation 2.1. The same holds if i is a delivery node. □

We remark that such a property does not hold outside the cluster structure; that is, in general, visiting the same node twice may be both needed for feasibility or simply be profitable.

We also observe that:

Observation 2.3. *The demand loaded in each node of a sequence of pickup nodes is irrelevant to both the feasibility and the cost of a cluster, as long as the sum of loaded demands remains constant. The same also applies to a delivery sequence.*

That is, each cluster can encode implicitly a potentially huge number of equivalent solutions. Also, let us extend the notion of k-split cycle introduced by Dror & Trudeau (1990) to clusters:

Definition 2.5. *Let $k > 1$ and $\tilde{N} = \{i_1, \ldots, i_k\} \subseteq N^+$ be given, $\tilde{N}$ is a* pickup *k-split cycle if there exist k clusters such that (a) nodes i_1 and i_k are visited in the same cluster, and (b) for every $t = 1, \ldots, k-1$, nodes i_t and i_{t+1} are visited in the same cluster.*

Similarly we can define a *delivery* k-split when $\tilde{N} \subseteq N^-$. For example, let us be given three routes $r_1 = (0, 1, 3, 2, 4, 0)$, $r_2 = (0, 3, 5, 6, 8, 0)$, and $r_3 = (0, 5, 7, 1, 8, 6, 0)$, where we suppose that odd numbers are pickup nodes, while even numbers are delivery nodes. In such example, there are two k-split cycles: one with $k = 3$ ($\tilde{N} = \{1, 3, 5\}$), and one with $k = 2$ ($\tilde{N} = \{6, 8\}$).

We prove that:

Theorem 2.1. *When c_{ij} costs and t_{ij} times satisfy triangular inequality, there always exists an optimal routing plan where no pickup (resp. delivery) k-split cycle is included.*

Proof. Let us be given an optimal solution having a k-split cycle $\tilde{N}$, and let us suppose w.l.o.g. that each node $i_t \in \tilde{N}$ is visited two distinct clusters, thus having two distinct loaded demands $\tilde{q}'_{i_t}$ and $\tilde{q}''_{i_t}$. For each node i_t we can move one unit of demand from $\tilde{q}'_{i_t}$ to $\tilde{q}''_{i_t}$ and from $\tilde{q}''_{i_t}$ to $\tilde{q}'_{i_{t+1}}$ without modifying the total amount of demand loaded in the clusters. Also, all $\tilde{q}'_{i_i}$ and $\tilde{q}''_{i_i}$ values remain constant besides $\tilde{q}'_{i_1}$ and $\tilde{q}''_{i_k}$ which are decreased and increased by one unit of demand, respectively. Therefore by repeating such operation $\tilde{q}'_{i_1}$ times, we obtain a loading $\tilde{q}'_{i_1} = 0$ and, because of Observation 2.1, a solution that is at least as good as the starting one, thus optimal. □

Indeed, Theorem 2.1 generalizes the Proposition by Dror and Trudeau for the SDVRPDror & Trudeau (1990).

2.2. Routes, clusters and loading patterns.

We now consider the particular subproblems arising when the routes are assumed to be given, and only a suitable loading/unloading plan needs to be found. Our main result is that by exploiting the clusters structure we are able to improve theoretical findings from the literature.

Theorem 2.2. *Given a set of feasible routes, one for each vehicle, the problem of finding a feasible loading plan can be solved in polynomial time.*

Proof. We provide a constructive proof, reducing to a maximum flow problem. Let us build a graph in which we have a source node s and a sink node t, a node f_i^+ for each pickup node i and a node f_j^- for each delivery node j, and two nodes p_{mk}^+ and p_{mk}^- for each cluster k of each vehicle m, corresponding to the subset of pickup and delivery nodes of cluster k, respectively. Let us add arcs from s to f_i^+ and from f_j^- to t with capacity d_i and d_j, respectively. For each pickup node i visited in a cluster k of vehicle m, add an arc from f_i^+ to p_{mk}^+ with infinite capacity. Similarly, for each delivery node j visited in a cluster k of vehicle m, add an arc from p_{mk}^- to f_j^- still with infinite capacity. From

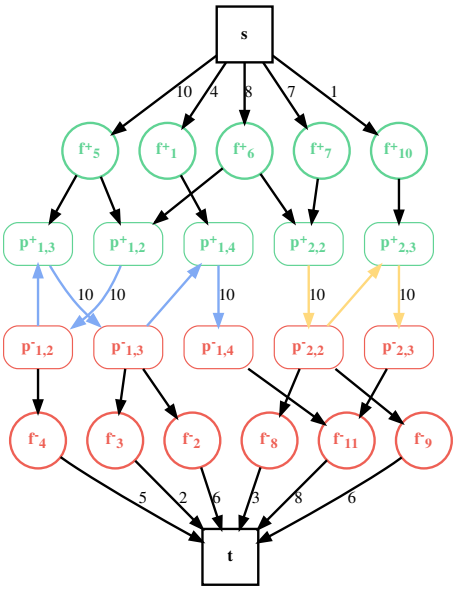

Figure 3: Resulting flow graph of the routes in the solution of Figure 1 assuming vehicles with capacity $Q = 10$. The labels on the arcs are the capacities of each arc (infinite capacities are omitted).

each node p^+_{mk} add an arc to p^-_{mk} with capacity Q, and from each node p^-_{mk} add an arc to p^+_{mk+1} with infinite capacity. An example of the graph is depicted in Figure 3.

A maximum flow solution on such a graph ensures that at most $\sum_{i \in N^+} d_i$ units can be loaded, and at most $\sum_{i \in N^-} d_i$ units unloaded. For each node, the units loaded (unloaded) are at most the demand of the node itself because of the limited capacity of ingoing arcs in f^+_i (outgoing arcs from f^-_j). Nodes p^+_{mk} and p^-_{mk} represent the load of the vehicle after pickups and deliveries, respectively. No vehicle is overloaded due to the limited capacity of arcs (p^+_{mk}, p^-_{mk}). Loading (unloading) is possible only at nodes corresponding to pickup (delivery) clusters, since they are the only ones connected to the source (depot). The flow passing from each node corresponding to pickup (delivery) cluster is at most Q, satisfying vehicles capacity constraints.

The quantities loaded and unloaded at nodes i and j of a cluster k of vehicle m are given by the flow on the arc (f^+_i, p^+_{mk}) and (p^-_{mk}, f^-_j), respectively. □

Thus, the reduction used in the proof of Theorem 2.2 can be used to *actually find* a loading plan. In particular,

Corollary 2.3. *If the flow reaching the depot t is less than the demands of pickup (delivery) nodes, then the starting assignment of nodes to clusters does not represent a feasible solution.*

This follows from the fact that some demands are not satisfied if the flow is less than their sum.

Furthermore, we remark that in our proof of Theorem 2.2, the pickup nodes (resp. delivery nodes) visited in a cluster are connected to the same p^+_{mk} (resp. p^-_{mk}) node, disregarding their order of visit in that cluster. In fact, the order of visits inside a cluster is irrelevant to build a loading plan. Formally, we prove the following:

Proposition 2.2. *When the set of nodes visited in each cluster of each vehicle is fixed, it is always possible to find a feasible loading plan in polynomial time, or to prove that no feasible loading plan exists.*

Our approach is indeed similar to the one presented in Casazza et al. (2018a) where, given a set of routes without loading quantities, a flow formulation is used to obtain such a missing information. However, our proof is different: from a theoretical point of view, our approach allows to build a smaller support graph, potentially yielding a better computational behaviour. In details, the graph of Casazza et al. (2018a) has two nodes for each customer node of the problem, while our graph has one node for each customer node, and two nodes for each cluster. Therefore, our graph has always a strictly smaller number of nodes, except in the extreme scenario in which exactly one pickup and one delivery node is visited in each cluster. In such scenario, the number of nodes of the two graphs is identical. Furthermore, our approach requires less information about the order of the nodes in the route. This makes it additionally suited to early detect the infeasibility of a branching node in a branch-and-bound approach.

3. Extended formulation

We now exploit the features of clusters to obtain a new formulation of the SPSDVRP. Let $K = \{1, \dots\}$ be the set of potential available clusters, and let

$$\Gamma = \left\{ (\bar{\mathrm{z}}^\gamma_{ij}, \bar{\mathrm{w}}^\gamma_i, \bar{\mathrm{q}}^\gamma_i, \bar{\mathrm{s}}^\gamma_i) \in \mathbb{B}^{|A|} \times \mathbb{B}^{|N_0|} \times \mathbb{N}_0^{|N|} \times \mathbb{B}^{|N_0|} \right\}$$

be the set of all feasible patterns γ of a cluster, where $\bar{z}^\gamma_{ij}$ is 1 if arc (i, j) is selected, and 0 otherwise, $\bar{w}^\gamma_i$ is 1 if node i is visited, and 0 otherwise, $\bar{q}^\gamma_i$ is the quantity of demand of node i loaded, and $\bar{s}^\gamma_i$ is 1 if node i is the first or the last node visited in the cluster, and 0 otherwise. We assume that there exist two types of clusters: clusters where customers nodes in N are visited, and clusters visiting the depot 0 only. Therefore, if $\bar{w}^\gamma_0 = 1$ then $\bar{\mathrm{z}} = \{0 \dots 0\}$, $\bar{\mathrm{q}} = \{0 \dots 0\}$, $\bar{s}^\gamma_0 = 1$, and $\bar{s}^\gamma_i = 0$ for all $i \in N$.

We introduce the following variables:

$$y^k_{m\gamma} = \begin{cases} 1 & \text{if pattern } \gamma \text{ is selected for cluster } k \text{ in vehicle } m \\ 0 & \text{otherwise} \end{cases}$$

$$x^k_{mij} = \begin{cases} 1 & \text{if arc } (i, j) \text{ is selected to link clusters } k \text{ and } k+1 \text{ in vehicle } m \\ 0 & \text{otherwise} \end{cases}$$

$f^k_m \geq 0$: amount of demand loaded on vehicle m after visiting nodes in cluster k

The SPSDVRP can be modelled as follows:

$$\min \sum_{m\in M}\sum_{k\in K}\sum_{(i,j)\in A} c_{ij}(x^k_{mij} + \sum_{\gamma\in\Gamma} \bar{z}^\gamma_{ij} y^k_{m\gamma}) \tag{1}$$

$$\text{s.t.} \sum_{m\in M}\sum_{k\in K}\sum_{\gamma\in\Gamma} \bar{q}^\gamma_i y^k_{m\gamma} = d_i \qquad \forall i \in N \tag{2}$$

$$f^k_m + \sum_{\gamma\in\Gamma}(\sum_{i\in N^+} \bar{q}^\gamma_i - \sum_{i\in N^-} \bar{q}^\gamma_i) y^{k+1}_{m\gamma} = f^{k+1}_m \qquad \forall m \in M, k \in K \tag{3}$$

$$f^k_m + \sum_{\gamma\in\Gamma}\sum_{i\in N^+} \bar{q}^\gamma_i y^{k+1}_{m\gamma} + Q\bar{s}^\gamma_0 y^k_{m\gamma} \le Q \qquad \forall m \in M, k \in K \tag{4}$$

$$\sum_{\gamma\in\Gamma} \bar{s}^\gamma_i y^k_{m\gamma} \le \sum_{j\in N^+_0} x^k_{mij} \qquad \forall m \in M, k \in K, i \in N^-_0 \tag{5}$$

$$\sum_{\gamma\in\Gamma} \bar{s}^\gamma_i y^k_{m\gamma} \le \sum_{j\in N^-_0} x^{k-1}_{mji} \qquad \forall m \in M, k \in K, i \in N^+_0 \tag{6}$$

$$\sum_{\gamma\in\Gamma} \bar{s}^\gamma_0 y^1_{m\gamma} = \sum_{k\in K:k>1}\sum_{\gamma\in\Gamma} \bar{s}^\gamma_0 y^k_{m\gamma} = 1 \qquad \forall m \in M \tag{7}$$

$$\sum_{(i,j)\in A} x^k_{mij} \le 1 \qquad \forall m \in M, k \in K \tag{8}$$

$$\sum_{(i,j)\in A} x^k_{mij} \le \sum_{(i,j)\in A} x^{k-1}_{mij} - \sum_{i\in N} x^{k-1}_{mi0} \qquad \forall m \in M, k \in K \tag{9}$$

$$\sum_{k\in K}\sum_{(i,j)\in A} t_{ij}(x^k_{mij} + \sum_{\gamma\in\Gamma} \bar{z}^\gamma_{ij} y^k_{m\gamma}) \le T \qquad \forall m \in M \tag{10}$$

$$y^k_{m\gamma} \in \mathbb{B} \qquad \forall m \in M, k \in K, \gamma \in \Gamma \tag{11}$$

$$x^k_{mij} \in \mathbb{B} \qquad \forall m \in M, k \in K, (i,j) \in A \tag{12}$$

$$0 \le f^k_m \le Q \qquad \forall m \in M, k \in K \tag{13}$$

The objective function (1) minimizes the cost of the routing plans. Constraints (2) ensure that the demand of each node is satisfied. Constraints (3) and (4) ensure consistency of the flow in the routing plan and impose that the capacity of the vehicle is never exceeded, respectively. Constraints (5) and (6) impose the use of linking arcs between each cluster. Constraints (7) force each vehicle to start and end its route at the depot. Constraints (8) and (9) impose that at most one linking arc is selected to connect two clusters and none is selected after the depot is visited, respectively. Finally, constraints (10) impose that each route does not exceed the time resource T.

3.1. An upper bound to the number of clusters.

Keeping the size of the set K of potential clusters small is of key importance to keep our formulation manageable. In fact, the number of clusters of an optimal routing plan is not known in advance, but it influences the number of variables of our formulation: therefore an unbounded number of clusters implies an unbounded number of variables. However, we can algorithmically infer bounds on it. We start by observing that:

Observation 3.1. *An upper bound n_{max} to the maximum number of nodes visited in a route can be obtained by solving a Resource Constrained Elementary Shortest Path Problem (RCESPP).*

In fact we obtain n_{max} by simply solving, in a preprocessing phase, a single RCESPP starting and ending in the depot, where each node in N has a unitary profit. Such a RCESPP involves a *single* time resource T that is consumed by t_{ij} when travelling arc (i, j).

Furthermore, the maximum number of visited nodes directly influences the maximum number of clusters of each vehicle:

Observation 3.2. *Each route has at most*

$$k_{max} = \left\lfloor \frac{n_{max}}{2} \right\rfloor + 2$$

clusters.

In fact, if we are given the number of nodes visited in a route we can obtain the maximum number of clusters of such a route by minimizing the number of nodes visited in each cluster, that is two nodes per cluster. Then we need two additional clusters for the starting and ending depot.

3.2. Valid inequalities.

To strengthen our formulation and reduce symmetries we add sets of valid inequalities. We start from observing that:

Observation 3.3. *Each node whose demand is partially collected in a selected cluster pattern γ is visited at least twice.*

We then add the following inequalities in our formulation:

$$\sum_{m \in M} \sum_{k \in K} \sum_{\gamma \in \Gamma} \frac{1 + \lfloor \bar{q}_i^{\gamma} / d_i \rfloor}{2} y_{m\gamma}^k \geq 1, \forall i \in N. \tag{14}$$

To enforce a minimum integer number of selected clusters in a solution, we add the inequality

$$\sum_{m \in M} \sum_{k \in K} \sum_{\gamma \in \Gamma} y_{m\gamma}^k \geq \left\lceil \frac{\sum_{i \in N^+} d_i}{Q} \right\rceil. \tag{15}$$

Finally, to reduce symmetries between vehicles, we sort vehicles by the number of their clusters:

$$\sum_{(i,j)\in A} x^k_{mij} \geq \sum_{(i,j)\in A} x^k_{m'ij}, \forall m \in M, m' = m+1, k \in K. \tag{16}$$

These inequalities are polynomial in number: we generate and add them to our formulation during preprocessing as well.

4. Algorithms

Straightly solving model (1) – (13), (14), (15), and (16) would be impractical due to the number of $y^k_{m\gamma}$ variables, that grows exponentially both in the number of nodes and in the range of demand values; therefore, we relax the integrality conditions and solve the so-called *Master Problem* (MP), that is continuous relaxation of such a model, by means of column generation (CG) techniques.

We start by solving a Restricted MP (RMP) involving a small initial set of columns (see Subsection 4.1), and then we use dual information to iteratively search for negative reduced cost variables by solving a pricing problem (see Subsections 4.2 and Subsection 4.3 for implementation details). If any negative reduced cost variable is found, it is added to the RMP and the CG process is repeated, otherwise the optimal RMP solution is optimal for the MP as well, and therefore the corresponding value is retained as a valid lower bound for the SPSDVRP. If the final RMP solution is integer, then it is also optimal for the SPSDVRP, otherwise we enter a search tree by performing branching operations (see Subsection 4.4) to find a proven global optimum. To speed up the pruning of non-optimal solutions, we also designed a rounding heuristic to find good primal bounds (see Subsection 4.5).

4.1. Initialization

We remark that, according to our definition of pattern variables $y^k_{m\gamma}$, if the depot is visited in a cluster ($\bar{w}^\gamma_0 = 1$), then no other node is visited in such a cluster ($\bar{w}^\gamma_i = 0$) for each $i \in N$. In turn, depot-only clusters are always needed to obtain a feasible solution.

Therefore, we first add to Γ a column $\hat{\gamma}$ where $z^{\hat{\gamma}}_{ij} = 0$ for all $(i,j) \in A$, $w^{\hat{\gamma}}_0 = s^{\hat{\gamma}}_0 = 1$, and $w^{\hat{\gamma}}_i = s^{\hat{\gamma}}_i = q^{\hat{\gamma}}_i = 0$ for all $i \in N$. Then, we populate the RMP with a variable $y^k_{m\hat{\gamma}}$ for each vehicle $m \in M$ and cluster $k \in K$.

As a positive side effect, we can skip the search for these columns during pricing.

Additionally, we exploit these initial columns to reduce symmetries in our formulation. In fact, we observe that:

Observation 4.1. *Let k_{min} be a lower bound to the minimum number of clusters required to perform all the pickups and deliveries, that is*

$$k_{min} = \left\lceil \frac{\sum_{i\in N^+} d_i}{Q} \right\rceil,$$

whenever $(m-1)(k_{max}-2) < k_{min}$ *we fix the variable encoding the depot-only cluster for vehicle* m *to* 1.

In fact, depot-only cluster variables for vehicle m are set to 0 if and only if, m is left unused; we leave it as an option only if the vehicles of index $m' < m$ are able to perform all the pickups and deliveries on their own.

4.2. Pricing problem

Due to our initialization method, the pricer can neglect clusters depot.

Let λ, μ, ζ, ν^+, ν^-, η, π, and θ be the dual variables of constraints (2), (3), (4), (5), (6), (10), (14), and (15), respectively.

First, since the values of ν^+ and ν^- are indexed by nodes in N^+ and N^-, respectively, let us simplify notation as

$$\nu_i = \begin{cases} \nu_i^+ & \text{if } i \in N^+ \\ \nu_i^- & \text{if } i \in N^- \end{cases}.$$

For each $\gamma \in \Gamma$, the reduced cost of variable $y_{m\gamma}^k$ is computed as

$$\begin{aligned} \sigma^\gamma = & \sum_{(i,j)\in A} c_{ij}\bar{z}_{ij}^\gamma - \sum_{(i,j)\in A} \eta^m t_{ij}\bar{z}_{ij}^\gamma - \sum_{i\in N} \pi_i \frac{1+\lfloor \bar{q}_i^\gamma/d_i \rfloor}{2} \\ & - \sum_{i\in N} \lambda_i \bar{q}_i^\gamma - \sum_{i\in N^+} \mu^{mk}\bar{q}_i^\gamma + \sum_{i\in N^-} \mu^{mk}\bar{q}_i^\gamma - \sum_{i\in N^+} \zeta^{mk}\bar{q}_i^\gamma \\ & - \sum_{i\in N} \nu_i^{mk}\bar{s}_i^\gamma - \theta. \end{aligned}$$

Now, let us aggregate coefficients as follows:

- $\alpha_{ij}^{mk} = c_{ij} - \eta^m t_{ij}$ is the cost of arc (i,j);
- $\beta_i^{mk} = \lambda_i + \mu^{mk} + \zeta^{mk}$ and $\beta_i^{mk} = \lambda_i - \mu^{mk}$ are either the profit or the cost for loading one unit of demand in pickup node $i \in N^+$ and delivery node $i \in N^-$, respectively.

The reduced cost σ^γ can be rewritten accordingly:

$$\sigma^\gamma = \sum_{(i,j)\in A} \alpha_{ij}^{mk}\bar{z}_{ij}^\gamma - \sum_{i\in N} \beta_i^{mk}\bar{q}_i^\gamma - \sum_{i\in N} \pi_i \frac{1+\lfloor \bar{q}_i^\gamma/d_i \rfloor}{2} - \sum_{i\in N} \nu_i^{mk}\bar{s}_i^\gamma - \theta. \quad (17)$$

We can finally formulate the pricing problem as follows:

$$\begin{aligned} \min_{m\in M, k\in K} & \sum_{(i,j)\in A} \alpha_{ij}^{mk} z_{ij} - \sum_{i\in N} \beta_i^{mk} q_i \\ & - \sum_{i\in N} \pi_i \frac{1+\lfloor q_i/d_i \rfloor}{2} - \sum_{i\in N} \nu_i^{mk} s_i - \theta & (18) \\ \text{s.t. } & w_i \le q_i \le d_i w_i & \forall i \in N \quad (19) \end{aligned}$$

$$\sum_{i \in \hat{N}} q_i \leq Q \qquad \forall \hat{N} \in \{N^+, N^-\} \quad (20)$$

$$\sum_{(i,j) \in A} z_{ij} = \sum_{j \in N^+} z_{ji} + s_i = w_i \qquad \forall i \in N^+ \quad (21)$$

$$\sum_{j \in N^-} z_{ij} + s_i = \sum_{(j,i) \in A} z_{ji} = w_i \qquad \forall i \in N^- \quad (22)$$

$$\sum_{i \in \hat{N}} s_i \leq 1 \qquad \forall \hat{N} \in \{N^+, N^-\} \quad (23)$$

$$\sum_{i \in N \setminus S, j \in S} (z_{ij} + z_{ji}) \geq w_u \qquad \forall S \subset N, |S| > 0, u \in S \quad (24)$$

$$z_{ij} \in \mathbb{B} \qquad (i,j) \in A \quad (25)$$

$$w_i \in \mathbb{B}, s_i \in \mathbb{B} \qquad i \in N \quad (26)$$

where variable z_{ij} is 1 if arc (i,j) is selected, and 0 otherwise, variable w_i is 1 if node i is visited in the cluster, and 0 otherwise, and variable s_i is 1 if node i is either the starting or the ending node of the cluster.

The objective function (18) targets at minimizing the reduced cost. Constraints (19) bound the amount of demand loaded at each node and impose that if a node is visited, at least one unit of its demand is loaded. Constraints (20) impose that both the pickup and delivery demands loaded do not exceed the capacity Q. Constraints (21) and (22) ensure that each visited node has exactly one ingoing and one outgoing arc. The only exceptions are the first pickup node and the last delivery node of the cluster, having no ingoing and no outgoing arc, respectively. Constraints (23) impose that there is at most one pickup node that starts the cluster and one delivery node that ends the cluster. Finally, (24) are subtour elimination constraints.

We observe that for a given solution to our pricing problem we may have three kinds of visited nodes:

integer nodes: those having their demands fully collected (i.e. $q_i = d_i$);

bridge nodes: those having exactly one unit of demand collected (i.e. $q_i = 1$).

fractional nodes: those having their demands fractionally collected (i.e. $1 < q_i < d_i$);

However, we prove that:

Theorem 4.1. *There always exists an optimal solution of* (18) – (26) *having at most one fractional pickup and one fractional delivery node.*

Proof. Assume by contradiction that there exists an optimal solution where all nodes are integer but i and j, which are pickup nodes and have both fractional demands q_i and q_j. Let $r = q_i + q_j$ be the capacity used by fractional demands of nodes i and j. Let us assume w.l.o.g. that node i is more efficient than node j, that is $\beta_i^{mk} > \beta_j^{mk}$. We can improve the contribution to the objective

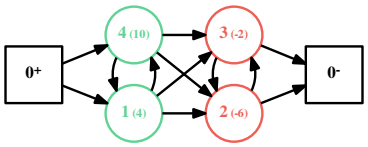

Figure 4: Example of auxiliary graph used to solve the pricing problem: pickup nodes are unreachable from the time the vehicle visits the first delivery node.

value (18) by setting $q_i = \min\{d_i, r\}$ and $q_j = \max\{0, r - d_i\}$, that is decreasing the capacity used by the less efficient node, and increasing the capacity used by the most efficient. The argument iteratively extends to solutions in which the number of nodes is higher than two, and symmetrically applies to pairs of fractional delivery nodes. □

Observation 4.2. *There always exists an optimal MP solution, in which each selected column contains at most one fractional node.*

In fact, each MP column can be found by solving the pricing problem, which in turn is allowed to generate only columns with at most one fractional node.

4.3. Pricing algorithms.

We remark that, given a pair of vehicle $m \in M$ and cluster $k \in K$, the pricing problem has two components: a routing one (finding sequences of nodes) and a loading one (finding amount of demand to serve in each of them). Despite the additional complexity given by the second component, we are able solve our pricing problem as a variant of the RCESPP.

To this aim let us define a directed graph $\tilde{G} = (\tilde{N}_0, \tilde{A})$, where $\tilde{N}_0$ is the set of nodes and $\tilde{A}$ is the set of arcs of the graph.

The set $\tilde{N}_0$ has one node for each pickup and delivery node of the original graph G, and two different copies of the depot, that is $\tilde{N}_0 = N \cup \{0^+, 0^-\}$.

The set $\tilde{A}$ instead has:

- arcs going from depot 0^+ to all pickup nodes $i \in N^+$;
- arcs going from pickup nodes $i \in N^+$ to all nodes $j \in N$ but the depots;
- arcs going from delivery nodes $i \in N^-$ to all nodes $j \in \tilde{N}_0 \setminus N^+$ but the pickup ones.

An example of such a graph is depicted in Figure 4.

Also, for each arc (i, j) we have a cost $\tilde{c}_{ij}$ such that

$$\tilde{c}_{ij} = \begin{cases} \nu_i^{mk}, & \text{if } i = 0^+ \vee j = 0^- \\ \alpha_{ij}^{mk}, & \text{otherwise} \end{cases} .$$

A single vehicle of capacity Q must travel from 0^+ to 0^- at minimum cost visiting each node at most once. When a node i is visited, at least one unit of

its demand is loaded into the vehicle, and for each unit of demand loaded we collect a profit β_i^{mk}. Also, if the demand of the node is fully loaded into the vehicle, we collect the full profit π_i, otherwise only $\pi_i/2$ is collected. The sum of pickup (resp. delivery) demands loaded into the vehicle must not exceed the capacity Q, however, such a capacity is always replenished when travelling from a pickup node to a delivery node.

We can prove the following.

Proposition 4.1. *Given a partial path from 0^+ to i, visiting a set of nodes $V \subseteq N$, the problem of finding an optimal pickup loading plan (resp. delivery loading plan) for such a partial path can be solved in pseudo-polynomial time.*

Proof. The problem of finding an optimal loading plan is equivalent to the problem of finding an optimal *Fractional Knapsack Problem with Penalties* (FKPP) Casazza & Ceselli (2014): let us be given a knapsack of capacity $\tilde{Q} = Q - |V|$, a set of items $I = V$, and for each item its weight $w_i = d_i - 1$, the profit $p_i = \beta_i^{mk}$ for each unit of weight packed, and the cost $c_i = \pi_i/2$ paid if item i is packed.

Since for each node in V, at least one unit of demand is collected, the knapsack has a fixed profit $\phi = \sum_{i \in I} p_i - c_i$. Solving a FKPP on such an instance returns a knapsack maximizing the profit

$$\phi^* = \sum_{i \in I} (p_i x_i - c_i y_i) + \sum_{i \in I} p_i - c_i,$$

where $0 \leq x_i \leq w_i$ is the quantity of item i packed into the knapsack and y_i is 1 if the item is packed. Items fully packed into the knapsack correspond to integer nodes, items that are not packed correspond to bridge nodes, and the fractional items correspond to fractional nodes. In such a way all nodes in V have at least one unit of demand collected, the capacity Q is never exceeded, for each node we collect at most d_i unit of demand. Also, for each instance of FKPP it always exists an optimal packing where only one item is fractionally packed Casazza & Ceselli (2014), and such a packing can be found in pseudo-polynomial time. □

Such a result allows us to focus on the routing component of the pricing problem: we devise a dynamic programming algorithm that is an extension of the well-known procedure for the RCESPP proposed in Righini & Salani (2006).

In our procedure a label $l = (i, c, V, r, f)$ corresponds to a partial path from 0^+ to i having cost c, visiting all nodes in V, with residual capacity r, and such that $f \in V \cup \{\varnothing\}$ is the fractional node, where $f = \varnothing$ means that no fractional node has been visited yet.

We now briefly describe the generic idea behind our algorithm, while we provide a more detailed description of the extension and dominance operations in the following paragraphs. Our algorithm proceeds as follows:

initialization: we start from a single label $l = (0^+, \theta, \emptyset, Q, \varnothing)$ and push it into a label queue;

selection: at each iteration we select the first label of the queue as the label to extend;

extension: given the selected label $l = (i, c, V, r, f)$, for each neighbour j of i we extend l creating three different labels, one where j is integer and its demand is fully collected, one where j is bridge and one unit of its demand is collected, and one where j is fractional;

dominance: we test each of the generated labels against the labels in the queue: if the new label is dominated by any of the labels in the queue, then it is discarded. Otherwise, it is pushed inside the queue;

termination: the algorithm ends when the queue is empty.

We remark that due to the structure of our graph, all paths generated using such an algorithm provide a sequence of pickup nodes followed by a sequence of delivery nodes.

At the end of our algorithm we search for the label ending in 0^- having minimum cost and exploit Observation 4.1 to retrieve an optimal loading plan corresponding to the route found. However, in our implementation we keep track of the loaded quantities during the extension operations.

We remark that our procedure is similar in nature to the one proposed in Desaulniers (2010) to solve the pricing problem in a branch-and-price approach for the Split Delivery Vehicle Routing Problem with Time Windows. This further highlights how our clusters may generalize nodes in SDVRPs.

We also mention that further modeling details might potentially fit in our framework, like quantity dependent loading and unloading time, without significantly change neither the formulation nor the structure of the pricing problem (e.g. by adapting the computation of β_i^{mk} values).

Label extension. Given a label $l = (i, c, V, r, f)$ we extend l to all neighbours j of i such that $j \notin V$, and we create three different labels where j is integer, bridge, and fractional. These three new labels are identical, except in the amount of demand $\tilde{q}$ that is loaded in each of the three cases, and in the tracking of the (single) current fractional node. In general, we the extension generates a label $l' = (j, c', V', r', f')$, where j is adjoined to the set of visited nodes $V' = V \cup \{j\}$, the cost of the partial path is $c' = c + \tilde{c}_{ij} - \beta_j^{mk}\tilde{q} - \pi_j(1 + \lfloor \tilde{q}/d_j \rfloor)/2$, and the residual capacity is decreased $r' = r - \tilde{q}$. Also, when travelling from a pickup node i to a delivery node j, the capacity is replenished, that is $r' = Q - \tilde{q}$, and the fractional node is reset $f' = \varnothing$. In any case, if $r' < 0$ the label is discarded. The differences between integer, bridge, and fractional extensions of node j are the following:

integer: the demand of j is fully loaded, that is $\tilde{q} = d_j$, and the fractional node of the label l' is the same as the one of l, that is $f' = f$;

bridge: the loaded demand $\tilde{q}$ is set to 1 and, similarly to the integer extension, the fractional node does not change, that is $f' = f$;

fractional: we set $\tilde{q} = 2$ and $f' = j$. We remark that the setting of $\tilde{q}$ is only a temporary lower bound to the vehicle capacity consumption: the actual

amount of loaded demand is set at its final value $\tilde{q} = \min\{d_j - 1, r\}$ only after the path is complete, and the residual vehicle capacity r is computed. It also follows that the fractional extension is performed only if the demand of the node is greater than 2, since otherwise the resulting label would be identical either to the bridge or to the integer one.

Also, both integer and fractional extensions are performed only if the profit β_j^{mk} is positive, otherwise if j is visited integer or fractional in an optimal solution, there always exists an equivalent optimal solution where it is visited as bridge.

Dominance rules. The drawback of our extension method is that we cannot collect the full profit of a fractional node until all pickup (resp. delivery) nodes are visited. Therefore, when checking the dominance of a given label with a fractional node, we have to ensure that such a label is dominated for every value of fractional loading.

Let us suppose that we are given two labels $l' = (i, c', r', V', f')$ and $l'' = (i, c'', r'', V'', f'')$, the necessary conditions for l' to dominate l'' are that l'' visits at least the same nodes of l' and that l' has less residual space that l', that is $V' \subseteq V''$ and $r' \geq r''$. For what concerns the cost instead, we can distinguish between 4 cases:

case 1 - $f' = f'' = \varnothing$: l' dominates l'' if l'' is more expensive, that is

$$c' \leq c''$$

case 2 - $f' = \varnothing \wedge f'' \neq \varnothing$: l' dominates l'' if l'' is more expensive even if the fractional demand is loaded at maximum, that is

$$c' \leq c'' - \beta_{f''}^{mk} \min\{d_{f''} - 3, r''\}$$

case 3 - $f' \neq \varnothing \wedge f'' = \varnothing$: l' dominates l'' if l'' is more expensive when l' uses the difference of residual spaces to load its fractional demand, that is:

$$c' - \beta_{f'}^{mk} \min\{d_{f'} - 3, r' - r''\} \leq c''$$

case 4 - $f' \neq \varnothing \wedge f'' \neq \varnothing$: l' dominates l'' if l'' is more expensive both when its fractional demand is loaded at minimum and at maximum value, that is case 3 and

$$c' - \beta_{f'}^{mk} \min\{d_{f'} - 3, r'', d_{f''} - 3 + (r' - r'')\} \leq c'' - \beta_{f''}^{mk} \min\{d_{f''} - 3, r''\}$$

Bidirectional algorithm. In order to improve the performance of our pricing algorithm, we exploit bidirectional search. Since each solution is composed of a sequence of pickup nodes, and a sequence of delivery nodes that can be treated independently, we run the algorithm on two different graphs, one composed by pickup nodes only, and one by delivery ones only. At the end, we obtain full solutions by joining the best labels of each pair of pickup and delivery nodes.

Heuristic pricer. To further speed up our CG procedure, we also include a heuristic version of our pricing algorithm where dominance rules are relaxed in such a way that the condition on the set V is ignored. The exact pricer is therefore run only when the heuristic pricer cannot find a column of negative reduced cost.

Pricer execution and insertion policy. In a strict CG approach, our pricing procedure should be computed for each pair of cluster $k \in K$ and vehicle $m \in M$, in order to find the column with minimum reduced cost. Instead, we found profitable from a computational point of view, to perform partial pricing. In details, we loop in order over the set of clusters and over the set of vehicles, stopping the loops as soon after we find a pair $\hat{k}$ and $\hat{m}$ yielding a negative reduced cost column. We then insert the corresponding column $\hat{\gamma}$ in Γ and add a variable $y^k_{m\hat{\gamma}}$ for each pair of cluster k and vehicle m, even if we have no guarantees that all those variables have a negative reduced cost.

4.4. Branching rules

When the optimal MP solution is fractional, and upper and lower bounds do not match, we check which integrality constraints are not satisfied and enforce them by exploring a search tree through branching rules.

First, we check whether an integer number of vehicles has been selected. Let $\tilde{y}^k_{m\gamma}$ be the value of a variable $y^k_{m\gamma}$ in a fractional solution to the MP, we search for the vehicle $\hat{m}$ having the most fractional depot node visited in the second cluster, that corresponds to a vehicle that does not leave the depot, that is

$$\hat{m} \in \operatorname*{argmin}_{m \in M} \left\{ \left| \sum_{\gamma \in \Gamma} \bar{w}^\gamma_0 \tilde{y}^2_{m\gamma} - \frac{1}{2} \right| \right\}.$$

If no fractional value is found, then we skip to the next branching rule, otherwise we perform binary branching: in the first child node we impose that vehicle $\hat{m}$ never leaves the depot by adding constraint

$$\sum_{\gamma \in \Gamma} \bar{w}^\gamma_0 y^2_{\hat{m}\gamma} \geq 1, \tag{27}$$

while in the second child we impose that the vehicle is always used, that is

$$\sum_{\gamma \in \Gamma} \bar{w}^\gamma_0 y^2_{\hat{m}\gamma} \leq 0. \tag{28}$$

We remark that both constraints have no impact on the resolution of the pricing problem, since all columns having depot only are included in the MP during an initialization phase.

When the number of vehicles is fixed, we check the integrality of the number of clusters used in each vehicle, searching for a pair of vehicle $\hat{m}$ and cluster $\hat{k}$

having the most fractional column with depot only, that is

$$(\hat{m}, \hat{k}) \in \underset{m \in M, k \in K}{\operatorname{argmin}} \left\{ \left| \sum_{\gamma \in \Gamma} \bar{w}_0^\gamma \tilde{y}_{m\gamma}^k - \frac{1}{2} \right| \right\}.$$

If such a fractional column exists, we perform binary branching and, similarly to the previous branching operation, we create two children where the depot is either fixed inside a cluster or excluded from such a cluster. In the first case we add to the MP constraint

$$\sum_{\gamma \in \Gamma} \bar{w}_0^\gamma y_{\hat{m}\gamma}^{\hat{k}} \geq 1, \tag{29}$$

while in the second one we add

$$\sum_{\gamma \in \Gamma} \bar{w}_0^\gamma y_{\hat{m}\gamma}^{\hat{k}} \leq 0. \tag{30}$$

Still, these constraints have no impact on our pricing problem.

Our last branching rule is devised to progressively assign nodes to clusters. We search for a tuple $(\hat{m}, \hat{k}, \hat{\imath})$ having the maximum fractional assignment of node $\hat{\imath}$ to cluster $\hat{k}$ of vehicle $\hat{m}$, that is

$$(\hat{m}, \hat{k}, \hat{\imath}) \in \underset{m \in M, k \in K, i \in N_0}{\operatorname{argmin}} \left\{ \left| \sum_{\gamma \in \Gamma} \bar{w}_i^\gamma \tilde{y}_{m\gamma}^k - \frac{1}{2} \right| \right\}.$$

If a fractional assignment is found we branch creating two children: one where we force $\hat{\imath}$ to be always visited by vehicle $\hat{m}$ in cluster $\hat{k}$ by adding constraint

$$\sum_{\gamma \in \Gamma} \bar{w}_{\hat{\imath}}^\gamma y_{\hat{m}\gamma}^{\hat{k}} \geq 1, \tag{31}$$

and one where we forbid such assignment

$$\sum_{\gamma \in \Gamma} \bar{w}_{\hat{\imath}}^\gamma y_{\hat{m}\gamma}^{\hat{k}} \leq 0. \tag{32}$$

From an implementation point of view, when such constraints are included in the MP, additional dual variables must be taken into account during the resolution of the pricing problem in the CG procedure. Constraint (31) and (32) origin a profit and a cost for visiting a node $\hat{\imath}$ in cluster $k \in K$ of vehicle $\hat{m}$, respectively. However, no modification is required, since in our pricing problem we already consider profits for visiting a node even if its demand is not profitable. Also, for what concerns constraint (32), to avoid the generation of infeasible columns and thus to speed up the CG procedure, we remove node $\hat{\imath}$ from the graph $\tilde{G}$ during the resolution of the dynamic programming procedure for cluster $\hat{k}$ and vehicle $\hat{m}$.

When no branching rule is applied, there may be some integrality constraints that are still not satisfied, for example the amount of demand loaded at each node, or the arcs selected in a route. However, we prove that:

Proposition 4.2. *If an integer assignment of nodes to cluster is given, we can stop the branching process and, in pseudo-polynomial time, either find a feasible solution or prove that such an assignment is infeasible.*

Proof. First given such an assignment, we can find a feasible loading plan by solving a maximum-flow problem as described in Theorem 2.2.

Then, for each vehicle we find a route of minimum cost by solving a particular variant of RCESPP where (a) the vehicle must visit all pickup nodes in a cluster before visiting a delivery node in the same cluster, and (b) the vehicle must visit all nodes in cluster k before the ones in cluster $k+1$.

Let us consider a single vehicle for which we are given the sets W^k of all the nodes visited in each cluster $k \in K$ by such a vehicle. We solve the problem of finding a minimum cost route using a dynamic programming algorithm where a label $l = (i, c, t, k, V)$ is a partial path from the depot 0 to node i in cluster k, of cost c and travelling time t, that visited all nodes in V. Our algorithm proceeds as follows:

initialization: we start from a single label $l = (0, 0, 0, 1, \emptyset)$ and we add such a label to a queue of labels;

label selection: at each iteration we extract the first label of the queue;

extension: we extend the selected label $l = (i, c, t, k, V)$ to a subset of neighbours j of i such that $j \notin V \cup \{0\}$, $t + t_{ij} \leq T$, and

$$\begin{cases} j \in N^+ \cap W^k \vee (j \in N^- \wedge V \cap W^k = N^+ \cap W^k) & \text{if } i \in N^+ \\ j \in N^- \cap W^k \vee (j \in (N_0 \setminus N^-) \cap W^{k+1} \wedge V \cap W^k = W^k) & \text{otherwise} \end{cases}$$

obtaining a new label $l' = (j, c', t', k', V')$ such that $c' = c + c_{ij}$, $t' = t + t_{ij}$, $V' = V \cup \{j\}$, and

$$k' = \begin{cases} k & \text{if } i \in N^+ \vee j \in N^- \\ k+1 & \text{otherwise} \end{cases}$$

If these these conditions are satisfied, the new label is pushed into the queue;

dominance: given two labels $l = (i, c, t, k, V)$ and $l' = (i, c', t', k, V')$, l dominates l' if $c \leq c'$, $t \leq t'$, $V' \subseteq V$;

termination: the algorithm ends when the labels queue is empty.

If our algorithm fails to find a feasible solution, then it means that the starting assignment of nodes to cluster does not provide a route that satisfies the time resource constraint. □

4.5. Rounding heuristic

To speed up the pruning of non-optimal branching nodes, we include in our framework a rounding heuristic that starts from a feasible fractional solution to the MP to find a feasible routing plan.

Let $\tilde{w}_i^{mk} = \sum_{\gamma \in \Gamma} \bar{w}_i^{\gamma} y_{m\gamma}^k$ be the fractional assignment of node i to cluster k of vehicle m. We select a threshold $0 < \rho < 1$ and then we round all $\tilde{w}_i^{mk}$ values, such that

$$w_i^{mk} = \begin{cases} 1 & \text{if } \tilde{w}_i^{mk} \geq \rho \\ 0 & \text{otherwise} \end{cases}, \forall m \in M, k \in K, i \in N_0$$

Given an integer assignment of nodes to cluster, we exploit the same properties of Proposition 4.2, and first we search for a feasible loading plan, and then for feasible routes. If any of these cannot be found, the integer assignment is discarded.

We found to be profitable to repeat such a procedure with different values of ρ, such that given the difference between the most and the least fractional assignment

$$\delta = \max_{m \in M, k \in K, i \in N_0} \tilde{w}_i^{mk} - \min_{m \in M, k \in K, i \in N_0} \tilde{w}_i^{mk},$$

ρ is selected in

$$\rho = \min_{m \in M, k \in K, i \in N_0} \tilde{w}_i^{mk} + \frac{k}{10}\delta, \forall k = 0, \ldots, 10.$$

We remark that if a feasible loading plan is not found for a given k, then no feasible loading plan can be found for $k' > k$.

5. Experimental analysis

We implemented our algorithms in C++, using the SCIP framework Achterberg (2009) version 4.0.0, while the LP subproblems were solved using the simplex algorithm implemented in CPLEX 12.6 CPLEX development team (2011): the framework automatically switches between primal and dual methods.

As a benchmark we considered the dataset used in Casazza et al. (2018a) and available at Casazza et al. (2018b), where each instance describes a randomly generated complete network with nodes located in the two-dimensional space $[-500, 500] \times [-500, 500]$, with the depot located at $(0, 0)$. Travelling costs c_{ij} are computed as the Euclidean distance between nodes i and j. Each node has a randomly generated integer demand between $[-10, 10]$, where positive values define pickup nodes, and negative values define delivery ones. We initially considered the same settings of Casazza et al. (2018a), where vehicle capacity Q is set to 10, and the number of available vehicles is $|M| = 5$. The name of each instance is in the form nXXqYYL, where XX is the number of nodes, YY is the vehicle capacity, and L identifies the network.

We remark that our dataset was generated by selecting those instances of Chemla et al. (2013) with up to 30 nodes and, in addition, a set of smaller instances was generated in Casazza et al. (2018a), where each instance with 20 nodes was split in two smaller instances of 10 nodes each, one with an upper case and one with a lower case letter at the end.

Since travelling times were not handled in earlier works, t_{ij} coefficients were not defined. We therefore proceeded as follows: instances in the dataset were considered in lexicographical order (assuming first instance to be successor of the last one); for each instance, the travelling times t_{ij} were set as the euclidean distance between nodes i and j in the *subsequent instance* according to the lexicographical order. This allowed us to obtain unrelated travel times and costs. We consider those distances to be in seconds, and then set the time resource $T = 3600$.

All tests have been conducted on a PC equipped with an Intel Core i7 6700K CPU clocked at 4.00GHz and 32 GB of RAM setting a time limit of 3 hours of computation for each run. All CPLEX options have been kept at default values, except for multi-threading support, which has been turned off.

We report that, as a preliminary benchmark, we designed a simple arc based formulation. However, it performed poorly, yielding timeouts already on instances with 10 nodes. Therefore, we did not further experiment with such an approach.

5.1. Algorithm profiling

To study the performance of our methodology, in a first round of tests we observed how the computing time was spent during the solving process.

In Table 1 we report for each instance the sum of the demands $\mathcal{D} = \sum_{i \in N} d_i$, the gap between upper and lower bound at the end of the computation, the number of branching nodes explored, the number of both CG iterations and generated columns, the percentage of time spent solving the LP subproblems and the pricing problem, and the computing time.

Instances with 30 nodes seem out of reach for our exact solution procedure (for three of them our algorithm was not even able to find a feasible upper bound), good duality gaps are consistently reached on instances with 20 nodes within our time limit, one instance being solved to proven optimality. All instances with 10 nodes can be solved in few minutes. These results are in line with previous computational experience on simpler SPSDVRP variants.

It is interesting to observe that on average 77% of the time is spent in computing the LP subproblems. An opposite computational behaviour is reported in Casazza et al. (2018a). This phenomenon proves that our findings are indeed very effective in speeding up a complex pricing algorithm, even if the price is the need of handling a more complex MP.

5.2. Comparison with unitary travelling time

In a second round of tests we evaluated the performance of our methods (BAP) on instances with unitary travelling time only. The lack of travelling

Instance	$\mathcal{D}$	gap (%)	#nodes	#CG iter. (#col.)	LP (%)	pricer (%)	time (s)
n10q10a	48	0.0	63	332 (2061)	78	16	6
n10q10A	52	0.0	49	210 (1374)	79	15	3
n10q10b	32	0.0	653	1897 (4167)	78	11	32
n10q10B	60	0.0	3404	8287 (23066)	72	10	361
n10q10c	44	0.0	32	180 (2456)	79	18	3
n10q10C	64	0.0	386	1644 (6264)	80	13	32
n10q10d	42	0.0	351	1780 (8167)	80	13	52
n10q10D	54	0.0	2048	4063 (5374)	74	12	77
n10q10e	58	0.0	1660	5225 (15714)	77	10	213
n10q10E	58	0.0	151	595 (2221)	82	13	10
n10q10f	36	0.0	413	1726 (5264)	72	19	29
n10q10F	50	0.0	310	1084 (3892)	76	16	17
n10q10g	34	0.0	217	914 (3907)	73	20	14
n10q10G	52	0.0	458	1671 (5088)	68	22	29
n10q10h	62	0.0	130	456 (2683)	86	10	8
n10q10H	48	0.0	504	1445 (3967)	80	11	26
n10q10i	60	0.0	187	717 (5515)	76	18	18
n10q10I	44	0.0	104	399 (2744)	86	10	6
n10q10j	42	0.0	368	1368 (3873)	81	11	27
n10q10J	56	0.0	542	1427 (3884)	79	13	23
n20q10A	88	0.0	20723	43663 (50694)	72	14	5720
n20q10B	80	6.4	38168	73678 (64517)	64	17	-
n20q10C	108	10.7	21735	46176 (47336)	77	14	-
n20q10D	118	19.5	53000	75520 (41578)	71	13	-
n20q10E	112	12.5	40122	64849 (42629)	76	11	-
n20q10F	90	2.3	22906	55809 (74420)	65	21	-
n20q10G	86	15.7	24257	57922 (65954)	72	15	-
n20q10H	100	8.2	34442	67485 (63219)	70	14	-
n20q10I	102	26.1	32865	55982 (46965)	76	11	-
n20q10J	92	8.4	46047	79919 (54906)	69	12	-
n30q10A	140	-	10705	25305 (51368)	71	23	-
n30q10B	124	120.9	8345	24430 (48504)	83	12	-
n30q10C	152	-	8089	19695 (50470)	79	17	-
n30q10D	146	96.6	12202	26351 (42652)	77	18	-
n30q10E	132	114.0	5923	16391 (47943)	85	12	-
n30q10F	130	-	72	504 (43973)	99	1	-
n30q10G	162	46.0	5416	14808 (45126)	84	13	-
n30q10H	136	85.7	10126	22254 (44287)	80	15	-
n30q10I	144	103.1	8470	19809 (39899)	83	13	-
n30q10J	138	37.8	7838	21193 (46494)	81	14	-

Table 1: Results for instances with up to 30 nodes, a travel time $T = 3600$, and 3 hours ot time limit.

time modelling was indeed one of the issues left open in Casazza et al. (2018a). The aim was therefore both to understand the overhead needed to handle our additional modelling flexibility, and to compare to the previous approach of Casazza et al. (2018a) (MVB), checking the price of the additional modelling freedom of our current approach.

To obtain a fair comparison we therefore set $T = 10$ and replaced each travelling time as $t_{ij} = 1$. This means that each vehicle can visit at most 9 customers before going back to the depot.

In Table 2 we report for each instance the sum of demands $\mathcal{D}$, the gap between upper and lower bound and the computing time of MVB, and the number of branching nodes, the number of CG iterations and columns, the gap, and the computing time of BAP.

We first compare the results of BAP on these simplified instances (last three columns of Table 2) and on their counterparts having full travelling time details (columns gap, #nodes and time of Table 1). As expected, explicitly including time details is actually making the problem more complex, especially in terms of number of branch-and-bound nodes that need to be explored to prove optimality.

Second, we compare the performance of BAP and MVB. For the set of instances with 20 and 30 nodes, BAP and MVB seem equivalent: neither algorithm can consistently solve these instances to proven optimality. MVB solves one instance more than BAP, but BAP is able to obtain a lower and an upper bound for all the instances, while MVB in three cases fails in finding any valid lower bound.

Restricting to those instances with 30 nodes where MVB finds a lower bound, MVB gaps are clearly smaller than BAP ones. Comparing the bounds obtained by both methodologies, we observed that BAP lower bounds were on average 8.6% worse than MVB ones, while upper bounds were 5.8% worse. Since the structure of these instances do not differ from that of smaller ones and the timeout is kept constant, we argue such a phenomenon to simply depend on a slower convergence process in optimizing larger instances through branching.

By looking at the instances with 10 nodes, BAP clearly outperforms MVB, with computing times which are orders of magnitude smaller. That is, despite being more flexible from the modelling point of view, our method proves to be more computationally effective than that of Casazza et al. (2018a).

To further investigate the differences between MVB and BAP, we report in Table 3 the effort required by both algorithms to compute a lower bound: for each instance we compute the best bound between the lower bound found by MVB and the lower bound at the root node of the branch-and-bound tree of BAP, and we report the gaps between the lower bounds the two algorithms and the best lower bound, and the corresponding computing time.

We can observe that when MVB terminates, its lower bound is the best, matching the theoretical expectation. The one found by BAP is up to 28.6% worse. However, for what concerns the computing time, BAP outperforms MVB by orders of magnitude: for instances with 10 nodes the computing time is negligible, and it is smaller than 10 seconds for all the instances with 20 and 30 nodes. Furthermore, for 3 instances with 30 nodes, MVB fails to find any

valid lower bound within 3 hours of computation. We can also observe that the computing time of MVB grows very quickly as the size of the instances increases, while that of BAP increases mildly.

Therefore, the advantage of BAP is the ability to find a lower bound in small computing time, even tough this feature leads to a significant reduction of the quality.

5.3. Increasing capacity

In a third round of experiments we evaluate the effect of optimizing instances with larger capacity values. Handling them was the second issue left open in Casazza et al. (2018a).

In Table 4 we report the results of the comparison between MVB and BAP on the same dataset but setting the capacity $Q = 20$.

The first observation is that both algorithms still solve all the instances with 10 items to proven optimality. However, with respect to the previous results in Table 2, while the average computing time required by BAP is substantially the same, we observe that MVB requires on average almost twice the computing time, and for instance n10q20C almost 3 hours.

The problems of MVB on instances with an increased capacity are confirmed by results with 20 and 30 nodes, where MVB fails in finding any valid lower bound for 16 instances out of 20, while BAP still solves to optimality 7 instances with 20 nodes.

Our conclusion is that BAP is less influenced by the value of the capacity of the vehicles, while MVB only works when such capacity is small.

5.4. Increasing time

It remains true that the performance of a pricing algorithm in routing problems are influenced by the overall length of the priced routes. To further test how our methodology in this direction, we repeated our tests doubling the value of time resource, that is setting $T = 7200$. In Table 5 we can observe that the results are very similar to the ones reported in Table, obtained by setting $T = 3600$: the number of instances solved to optimality is unchanged, and also the computing times and the number of branching nodes, although being different for the single instances, is on average very similar. Indeed, such a robustness comes *by design*: the value of T might influence the number of clusters available in each route, but not the length of the single cluster. That is, the complexity of the pricing problem (which is the most critical point in CG approaches like ours) is not affected by T.

6. Conclusions

Our investigation on the SPSDVRP led first of all to an interesting structural insight: optimal solutions always exist, having a particular structure in clusters; by making such a structure explicit, we are able not only to cast the SPSDVRP as a generalization of well known problems in transportation, but

also to generalize and extend the theoretical properties which are known for them. As a representative example, we proved that a cluster in SPSDVRP (that is a concatenation of a single sequence of pickup operations and a single sequence of delivery operations) generalizes a single node in a SDVRP, keeping at the same time many of the corresponding theoretical properties.

We have also proved that these properties can be exploited to obtain effective pricing algorithms, and full SPSDVRP exact algorithms as a consequence.

Even if the size of instances that can be consistently solved to proven optimality is still far from that of real networks, our experimental results revealed that our approach outperforms previous attempts from the literature in three regards. First, it offers more modeling flexibility: instances in which travel times are considered explicitly can be optimized with limited additional effort. Second, our algorithms are less affected by instance data, and in particular by the capacity of the vehicle. The width of the planning horizon shows also to have no computationally relevant effect. Third, it is faster on small instances and it keeps being able to provide lower bounds in reasonable computing time on larger ones.

Overall, our results indicate that although it is common practice in the design of column generation algorithms for routing problems to price full routes, thereby aiming for the best dual bound at the cost of solving a complex pricing problem (like in Casazza et al. (2018a)), alternative approaches exploiting decomposition results might be promising.

Acknowledgements

Partially funded by Regione Lombardia, grant agreement n. E97F17000000009, Project AD-COM. The authors wish to thank two anonymous reviewers for their insightful comments on the paper.

References

Achterberg, T. (2009). Scip: solving constraint integer programs. *Mathematical Programming Computation*, *1*, 1–41.

Alvarez-Valdes, R., Belenguer, J., Benavent, E., Bermudez, J., Muñoz, F., Vercher, E., & Verdejo, F. (2016). Optimizing the level of service quality of a bike-sharing system. *Omega*, *62*, 163 – 175.

Baldacci, R., Bartolini, E., & Mingozzi, A. (April 2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, *59*, 414–426.

Battarra, M., Cordeau, J.-F., & Iori, M. (2014). Chapter 6: Pickup-and-delivery problems for goods transportation. In P. Toth, & D. Vigo (Eds.), *Vehicle Routing* (pp. 161–191). doi:10.1137/1.9781611973594.ch6.

Bulhões, T., Subramanian, A., Erdoğan, G., & Laporte, G. (2018). The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, *264*, 508 – 523.

Casazza, M., & Ceselli, A. (2014). Mathematical programming algorithms for bin packing problems with item fragmentation. *Computers & Operations Research*, *46*, 1 – 11.

Casazza, M., Ceselli, A., Chemla, D., Meunier, F., & Wolfler Calvo, R. (2018a). The multiple vehicle balancing problem. *Networks*, *72*, 337–357.

Casazza, M., Ceselli, A., Chemla, D., Meunier, F., & Wolfler Calvo, R. (2018b). Multiple vehicle balancing problem instances. URL: `http://casazza.di.unimi.it/files/papers/spsdvrp.tar.gz`.

Casazza, M., Ceselli, A., & Wolfler Calvo, R. (2018c). A branch and price approach for the split pickup and split delivery vrp. *Electronic Notes in Discrete Mathematics*, *69*, 189 – 196. Joint EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization (EURO/ALIO 2018).

Chemla, D., Meunier, F., & Wolfler Calvo, R. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, *10*, 120–146.

CPLEX development team (2011). *IBM Ilog CPLEX Optimization Studio: CPLEX User's Manual - Version 12 Release 6*. Technical Report IBM corp.

Desaulniers, G. (2010). Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, *58*, 179–192.

Di Gaspero, L., Rendl, A., & Urli, T. (2013). A Hybrid ACO+CP for Balancing Bicycle Sharing Systems. In M. Blesa, C. Blum, P. Festa, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics* (pp. 198–212). Berlin, Heidelberg: Springer Berlin Heidelberg.

Dror, M., & Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics*, *37*, 383–402.

Espegren, H., Kristianslund, J., Andersson, H., & Fagerholt, K. (2016). The Static Bicycle Repositioning Problem - Literature Survey and New Formulation. In A. Paias, M. Ruthmair, & S. Voß (Eds.), *Computational Logistics* (pp. 337–351). Cham: Springer International Publishing.

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

Gschwind, T., Irnich, S., Rothenbächer, A.-K., & Tilk, C. (2018). Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems. *European Journal of Operational Research*, *266*, 521 – 530.

Laporte, G., Meunier, F., & Wolfler Calvo, R. (2015). Shared mobility systems. *4OR*, *13*, 341–360.

Pessoa, A., Sadykov, R., Uchoa, E., & Vanderbeck, F. (2019). A generic exact solver for vehicle routing and related problems. In A. Lodi, & V. Nagarajan (Eds.), *Integer Programming and Combinatorial Optimization* (pp. 354–369). Cham: Springer International Publishing.

Rainer-Harbach, M., Papazek, P., Hu, B., & Raidl, G. (2013). Balancing Bicycle Sharing Systems: A Variable Neighborhood Search Approach. In M. Middendorf, & C. Blum (Eds.), *Evolutionary Computation in Combinatorial Optimization* (pp. 121–132). Berlin, Heidelberg: Springer Berlin Heidelberg.

Rainer-Harbach, M., Papazek, P., Raidl, G., Hu, B., & Kloimüllner, C. (2015). PILOT, GRASP, and VNS approaches for the static balancing of bicycle sharing systems. *Journal of Global Optimization*, *63*, 597–629.

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

Righini, G., & Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, *3*, 255–273.

Instance	$\mathcal{D}$	MVB			BAP		
		gap (%)	time (s)	#nodes	#CG iter. (#col.)	gap (%)	time (s)
n10q10a	48	0.0	82	115	565 (2336)	0.0	5
n10q10A	52	0.0	35	15	94 (940)	0.0	1
n10q10b	32	0.0	63	117	485 (1855)	0.0	3
n10q10B	60	0.0	77	100	419 (1675)	0.0	3
n10q10c	44	0.0	34	1	23 (817)	0.0	0
n10q10C	64	0.0	4701	429	1572 (4571)	0.0	14
n10q10d	42	0.0	96	274	1414 (3809)	0.0	12
n10q10D	54	0.0	426	1017	2704 (5003)	0.0	28
n10q10e	58	0.0	241	980	3261 (7412)	0.0	41
n10q10E	58	0.0	204	229	940 (3385)	0.0	10
n10q10f	36	0.0	64	689	2778 (6329)	0.0	30
n10q10F	50	0.0	44	17	83 (1341)	0.0	1
n10q10g	34	0.0	67	101	381 (1840)	0.0	2
n10q10G	52	0.0	207	39	200 (2944)	0.0	2
n10q10h	62	0.0	1806	201	731 (2518)	0.0	6
n10q10H	48	0.0	615	233	756 (3155)	0.0	7
n10q10i	60	0.0	2923	332	1119 (4115)	0.0	11
n10q10I	44	0.0	105	252	871 (3293)	0.0	7
n10q10j	42	0.0	87	245	1088 (3988)	0.0	11
n10q10J	56	0.0	240	174	620 (2902)	0.0	4
n20q10A	88	0.0	981	8983	20115 (19648)	0.0	681
n20q10B	80	0.0	9951	67622	114946 (72724)	0.5	-
n20q10C	108	3.0	-	60173	109649 (77490)	1.5	-
n20q10D	118	0.0	518	98724	133953 (50723)	14.0	-
n20q10E	112	6.7	-	100544	147071 (63372)	8.4	-
n20q10F	90	0.0	771	1273	4710 (12099)	0.0	194
n20q10G	86	7.3	-	61816	122950 (88221)	6.6	-
n20q10H	100	12.4	-	30492	57759 (47093)	0.0	3266
n20q10I	102	8.2	-	76573	119861 (67279)	7.2	-
n20q10J	92	0.0	684	21418	42901 (35735)	0.0	2038
n30q10A	140	8.1	-	38785	76673 (62522)	31.4	-
n30q10B	124	9.0	-	47955	100053 (82084)	43.5	-
n30q10C	152	-	-	46006	80907 (57063)	47.9	-
n30q10D	146	19.2	-	42116	80969 (66967)	26.8	-
n30q10E	132	9.8	-	46798	88300 (65238)	19.1	-
n30q10F	130	8.7	-	30131	68089 (66979)	19.4	-
n30q10G	162	10.6	-	53247	89268 (57332)	26.8	-
n30q10H	136	16.0	-	49109	91616 (65381)	30.3	-
n30q10I	144	-	-	41815	80692 (65595)	18.8	-
n30q10J	138	-	-	32580	71382 (77218)	23.3	-

Table 2: Results for the comparison between BAP and MVB with instances with up to 30 nodes and 3 hours ot time limit.

Instance	$\mathcal{D}$	MVB		BAP	
		gap (%)	time (s)	gap (%)	time (s)
n10q10a	48	0.0	24	3.5	0
n10q10A	52	0.0	24	8.3	0
n10q10b	32	0.0	26	9.1	0
n10q10B	60	0.0	37	3.2	0
n10q10c	44	0.0	23	0.0	0
n10q10C	64	0.0	221	8.8	0
n10q10d	42	0.0	23	3.7	0
n10q10D	54	0.0	52	22.5	0
n10q10e	58	0.0	67	5.6	0
n10q10E	58	0.0	39	10.8	0
n10q10f	36	0.0	19	11.5	0
n10q10F	50	0.0	30	3.4	0
n10q10g	34	0.0	22	13.3	0
n10q10G	52	0.0	33	4.6	0
n10q10h	62	0.0	70	9.2	0
n10q10H	48	0.0	58	5.0	0
n10q10i	60	0.0	307	23.3	0
n10q10I	44	0.0	27	9.7	0
n10q10j	42	0.0	32	9.5	0
n10q10J	56	0.0	215	28.6	0
n20q10A	88	0.0	286	11.6	2
n20q10B	80	0.0	363	17.8	1
n20q10C	108	0.0	454	8.2	2
n20q10D	118	0.0	388	19.4	1
n20q10E	112	0.0	2492	15.9	2
n20q10F	90	0.0	333	7.4	2
n20q10G	86	0.0	7349	14.1	2
n20q10H	100	0.0	594	12.8	1
n20q10I	102	0.0	3409	16.1	1
n20q10J	92	0.0	620	21.8	1
n30q10A	140	0.0	3239	18.5	8
n30q10B	124	0.0	5582	22.1	6
n30q10C	152	-	-	0.0	6
n30q10D	146	0.0	7715	14.3	7
n30q10E	132	0.0	2869	12.3	6
n30q10F	130	0.0	7928	13.7	8
n30q10G	162	0.0	1569	15.2	7
n30q10H	136	0.0	6639	23.7	8
n30q10I	144	-	-	0.0	7
n30q10J	138	-	-	0.0	7

Table 3: Gap from the best lower bound for both BAP and MVB with instances with up to 30 nodes and 3 hours ot time limit.

Instance	$\mathcal{D}$	MVB			BAP		
		gap (%)	time (s)	#nodes	#CG iter. (#col.)	gap (%)	time (s)
n10q20a	48	0.0	77	29	173 (1258)	0.0	2
n10q20A	52	0.0	487	53	339 (1783)	0.0	4
n10q20b	32	0.0	90	85	317 (1386)	0.0	3
n10q20B	60	0.0	1105	95	389 (1901)	0.0	5
n10q20c	44	0.0	34	13	102 (1439)	0.0	2
n10q20C	64	0.0	10731	33	180 (1936)	0.0	2
n10q20d	42	0.0	51	47	349 (2256)	0.0	4
n10q20D	54	0.0	109	234	827 (3779)	0.0	13
n10q20e	58	0.0	702	41	242 (2494)	0.0	4
n10q20E	58	0.0	143	2	13 (679)	0.0	0
n10q20f	36	0.0	143	811	3297 (8802)	0.0	66
n10q20F	50	0.0	72	3	52 (1267)	0.0	1
n10q20g	34	0.0	136	115	420 (1562)	0.0	4
n10q20G	52	0.0	3754	25	154 (1640)	0.0	2
n10q20h	62	0.0	1278	123	619 (3124)	0.0	9
n10q20H	48	0.0	1343	244	892 (3417)	0.0	14
n10q20i	60	0.0	1548	203	734 (3440)	0.0	13
n10q20I	44	0.0	437	275	946 (3328)	0.0	12
n10q20j	42	0.0	46	57	308 (2013)	0.0	4
n10q20J	56	0.0	521	145	521 (2684)	0.0	7
n20q20A	88	-	-	16537	42367 (44661)	0.0	4356
n20q20B	80	0.0	10789	31047	65707 (61346)	0.0	8780
n20q20C	108	-	-	23195	70329 (80948)	0.0	10762
n20q20D	118	-	-	41670	83232 (70226)	0.3	-
n20q20E	112	-	-	3192	10195 (19967)	0.0	730
n20q20F	90	0.0	8358	758	3394 (9063)	0.0	621
n20q20G	86	0.0	10724	28438	79091 (87554)	0.2	-
n20q20H	100	0.0	6935	4069	14268 (21644)	0.0	974
n20q20I	102	-	-	49257	90062 (67084)	3.3	-
n20q20J	92	-	-	5960	16541 (20662)	0.0	1038
n30q20A	140	-	-	17768	43914 (48957)	20.5	-
n30q20B	124	-	-	17076	46152 (54320)	18.5	-
n30q20C	152	-	-	21411	48708 (48615)	16.6	-
n30q20D	146	-	-	18075	47214 (54889)	18.9	-
n30q20E	132	-	-	18090	55855 (440568)	19.5	-
n30q20F	130	-	-	17899	46839 (51898)	12.4	-
n30q20G	162	-	-	15661	42264 (50714)	25.9	-
n30q20H	136	-	-	988	4080 (11806)	19.8	-
n30q20I	144	-	-	8162	21687 (30082)	45.5	-
n30q20J	138	-	-	433	2031 (10205)	53.5	-

Table 4: Results obtained with an increased capacity $Q = 20$ with instances with up to 30 nodes and 3 hours ot time limit.

Instance	$\mathcal{D}$	gap (%)	#nodes	#CG iter. (#col.)	LP (%)	pricer (%)	time (s)
n10q10a	48	0.0	40	213 (2323)	75	21	5
n10q10A	52	0.0	195	750 (3658)	75	18	16
n10q10b	32	0.0	1091	3129 (6779)	78	10	60
n10q10B	60	0.0	843	2670 (7055)	72	18	65
n10q10c	44	0.0	11	99 (2400)	77	20	3
n10q10C	64	0.0	1314	4185 (8702)	78	12	120
n10q10d	42	0.0	103	608 (3144)	82	13	12
n10q10D	54	0.0	1306	3107 (5664)	76	11	66
n10q10e	58	0.0	3021	8453 (10809)	74	12	230
n10q10E	58	0.0	524	1817 (4699)	73	17	34
n10q10f	36	0.0	225	1093 (5233)	66	24	28
n10q10F	50	0.0	157	621 (4825)	74	20	15
n10q10g	34	0.0	59	272 (1581)	81	14	4
n10q10G	52	0.0	19	140 (2860)	54	43	4
n10q10h	62	0.0	227	793 (3017)	75	19	16
n10q10H	48	0.0	522	1828 (6280)	74	18	46
n10q10i	60	0.0	159	580 (3605)	73	19	11
n10q10I	44	0.0	101	391 (2582)	78	17	8
n10q10j	42	0.0	196	916 (4205)	72	22	23
n10q10J	56	0.0	139	551 (3007)	77	14	7
n20q10A	88	3.4	32137	62212 (53283)	70	16	-
n20q10B	80	8.1	31827	63822 (62006)	67	15	-
n20q10C	108	1.6	25477	54790 (51568)	72	17	-
n20q10D	118	20.0	30483	48096 (39405)	78	12	-
n20q10E	112	14.3	31724	53827 (38488)	80	9	-
n20q10F	90	0.0	3132	9547 (27937)	76	17	1864
n20q10G	86	14.2	26739	65065 (65389)	66	18	-
n20q10H	100	2.4	35410	68293 (54280)	72	13	-
n20q10I	102	13.5	24598	46629 (45527)	77	13	-
n20q10J	92	4.2	30349	57840 (51243)	75	12	-
n30q10A	140	56.6	2101	7483 (53537)	81	17	-
n30q10B	124	78.2	9894	29683 (51267)	78	16	-
n30q10C	152	80.0	4667	14176 (55870)	79	18	-
n30q10D	146	69.2	3405	9608 (36575)	90	9	-
n30q10E	132	55.2	5257	16331 (43098)	87	9	-
n30q10F	130	44.3	2737	9957 (42245)	86	11	-
n30q10G	162	45.0	4487	13690 (48880)	83	14	-
n30q10H	136	54.7	7144	19630 (47169)	80	15	-
n30q10I	144	49.7	6519	18752 (43805)	84	12	-
n30q10J	138	31.8	5240	17220 (49060)	80	14	-

Table 5: Results for instances with up to 30 nodes, a travel time $T = 7200$, and 3 hours ot time limit.