# A Branch-and-Cut algorithm for the split-demand one-commodity pickup-and-delivery travelling salesman problem ⋆

Hipólito Hernández-Pérez[a], Juan José Salazar-González[a]

[a]*DMEIO, Facultad de Ciencias, Universidad de La Laguna, 38200 La Laguna, Tenerife, Spain*

**Abstract**

This paper deals with the problem of designing a minimum-cost route for a capacitated vehicle moving a commodity between a set of customers, allowing two features uncommon in the pickup-and-delivery literature. One feature is that a customer accepts to be visited several times. It allows splitting a customer demand. The other feature is that a customer may be used as an intermediate location to collect and deliver commodity temporarily. The problem is called Split-Delivery One-Commodity Pickup-and-Delivery Travelling Salesman Problem, and finds applications in bike sharing systems where a single vehicle moves bikes between bike stations of a city district during the night to set the network to an initial configuration.

The paper proposes a new branch-and-cut algorithm to find optimal solutions. A master problem solves a relaxed Mixed Integer Programming model, i.e. a model allowing all feasible solutions but also some invalid ones. A subproblem checks the feasibility of the master solutions and generates valid cuts when they are infeasible. Computational results on benchmark instances demonstrate the good performance of the algorithm compared with others in the literature. In particular, it solves benchmark instances with 60 customers that were unsolved.

*Keywords:* Vehicle Routing Problem, Split Demand, Pickup-and-Delivery, One-Commodity, Bike Rebalance, Bike Shared System.

⋆This work has been partially supported by the research project MTM2015-63680-R (MINECO/FEDER) and ProID2017010132 (Gobierno de Canarias/FEDER).

*Email addresses:* hhperez@ull.es (Hipólito Hernández-Pérez), jjsalaza@ull.es (Juan José Salazar-González)

## 1. Introduction

The well-known Travelling Salesman Problem (TSP) looks for a minimum-cost circuit for a single uncapacitated vehicle visiting $n$ locations to move one unit of a commodity from one location (called pickup customer) to each other (called delivery customers). The One-Commodity Pickup-and-Delivery TSP (1PDTSP) generalizes the TSP by considering a capacitated vehicle, several pickup customers, and general customer demands. The Split-Demand 1PDTSP (SD1PDTSP) is a further generalization where the circuit is allowed to serve a customer with several visits. It is also a generalization of the well-known Capacitated Vehicle Routing Problem (CVRP), aimed at designing the route for a capacitated vehicle to deliver a commodity from one pickup location (called depot) to all the others (called customers). In the CVRP the depot can be visited at most a given number (the fleet size) and each customer can be visited at most once. The Split-Delivery CVRP (SDVRP) extends the CVRP by allowing several visits to a customer, thus it is a particular case of the SD1PDTSP.

An application of the SD1PDTSP arises in the context of a self-service bike-sharing system, which appears in many large and medium sized cities around the world. Every night a vehicle with limited load capacity has to reallocate the bicycles between stations in a district of a city. The logistic problem aims at finding a route for the vehicle with a minimum travel cost to restore the initial configuration of the system (see, e.g., Raviv et al. [15] and Dell'Amico et al. [5]).

More precisely, the SD1PDTSP is defined as follows. Let us consider a finite set of locations. One location represents a depot, which will be the initial and final location in the route. The other locations represent customers. The travel distances (or costs) between the locations are assumed to be known. Each customer is associated with an initial inventory of a commodity (e.g. bicycles at the end of a day) and with a desired inventory (e.g. the bicycles that are expected to be available early in the next morning). The difference between the two inventory levels is the customer demand, which may be positive or negative depending whether commodity is initially in deficiency or excess, respectively. Customers with positive

demands correspond to delivery locations, and customers with negative demands correspond to pickup locations. We assign a demand to the depot so that the sum of demands over all locations is zero. This allows to consider the depot as any customer when designing the route. In addition, each customer (also the depot) is associated with a capacity, representing the maximum amount of inventory that can be store at any time (e.g. the maximum number of bicycles that can be stored in that location). There is *one* vehicle with a given load capacity that must visit all customers, starting from and ending at the depot, through a route to move the commodity and satisfy all the customer demands without violating any capacity. The vehicle may partially satisfy a customer in a visit, so several visits to a location are allowed. Other important features to properly define the problem in this paper are the following:

1. The number of visits to each customer is upper limited. This assumption can be relaxed by simply considering a large number. However, in practice it is reasonable that a customer does not want to be disturbed many times. When this limit is one for each customer (also the depot), we refer to the split-forbidden variant of the problem, i.e., the SD1PDTSP becomes the 1PDTSP. Due to this upper limitation and to the vehicle capacity constraint, checking whether a feasible SD1PDTSP solution exists is an NP-complete problem.
2. Customers with zero demand must be visited. This assumption is motivated by a potential necessity of inspecting all locations, no matter the demand, It allows us to keep the TSP structure in the problem definition. Still, it is easy to adapt the approach in this paper to relax this assumption.
3. The initial load of the vehicle is unfixed. We do not assume a given location from which the vehicle must leave empty or full. The target is to satisfy the customer demands (in one or several visits) without violating the vehicle and customer capacity limitations. It implies that the depot has not a predefined inventory level to upper bound (or fix) the load of the vehicle when starting the route. The initial load of the vehicle can be seen as the initial inventory level at the depot that must be computed when solving

the SD1PDTSP, and it is easy to adapt the approach in this paper to bound or fix this level if desired.

4. Each customer may be used as an intermediate location to collect and deliver units temporarily. In other words, since several visits to a location are allowed, the vehicle could deliver some units of the commodity in a visit, and collect them later in another visit. Similarly some available units can be collected in a first visit and returned back in a second visit. This feature is called *preemption*, depends on the customer capacities, and may allow routes with smaller costs. Again, the approach in this paper is easily adaptable to relax this assumption and forbid preemption on some or all locations.

The 1PDTSP is the split-forbidden variant of the SD1PDTSP, and was introduced by Hernández-Pérez and Salazar-González [9], together with a branch-and-cut algorithm capable of solving to optimality instances with up to 60 customers. Hernández-Pérez and Salazar-González [10] describe two heuristic algorithms to deal with larger 1PDTSP instances. Other heuristic algorithms for the 1PDTSP are in Zhao et al. [17] and Mladenović et al. [12]. The first article proposes a *genetic algorithm* which uses several local-search procedures and a pheromone-based crossover operator. The second article describes a *general variable neighbourhood search* which combines extended neighbourhood structures from the TSP. This procedure is currently the best heuristic algorithm for solving large size 1PDTSP instances, both in computational time and solution quality.

Salazar-González and Santos-Hernández [16] introduce the SD1PDTSP, present a flow formulation, and describe a branch-and-cut algorithm based on Benders' Decomposition and tested on benchmark instances with up to 50 customers. An interesting observation from that article is that, on the used benchmark instances, a customer is rarely visited more than three times by an optimal route. Another interesting remark is that the complexity of solving an SD1PDTSP instance is similar with or without preemption. The complexity depends hardly on the vehicle capacity, being the problem more difficult when this capacity is smaller.

Chemla et al. [3] introduce another problem called *Single Vehicle One-commodity Ca-*

*pacitated Pickup and Delivery Problem* (SVOCPDP), very similar to the SD1PDTSP. The difference is in the initial load of the vehicle when leaving the depot, which in the SVOCPDP is assumed to be zero (the depot is assumed to have no commodity when the route starts). Chemla et al. [3] propose effective upper and lower bounding procedures, but did not implement an exact approach. The upper bounding procedure is a tabu search algorithm, the lower bounding procedure is a branch-and-cut algorithm to solve a relaxed model, and the procedures were tested on instances with up to 100 customers. Erdogan et al. [7] introduce another variant called *Static Bicycle Rebalancing Problem* (SBRP), where the initial load of the vehicle is unfixed but upper limited by the initial inventory level at the depot. Erdogan et al. [7] propose a branch-and-cut algorithm using Combinatorial Benders' cuts solving instances with up to 60 customers, and extended their algorithm to the non-preemptive SBRP, reported in the article to be significantly harder than the SBRP. Cruz et al. [4] describe an iterated local search approach to find heuristic solutions of the SBRP, tested on instances with up to 100 customers. SVOCPDP and SBRP have no limit on the maximum number of visits to a customer, and customers with zero demand are not required to be visited.

We emphasize that a customer with positive demand is a customer requiring product (i.e. a delivery customer) in our paper. This convention is in coherence with other articles in the vehicle routing literature (see e.g. Archetti and Speranza [2]). Whereas, Chemla et al. [3] and Erdogan et al. [7] use the opposite convention where positive demand means pickup.

There are several works related to balance bike-share systems with more than one vehicle. Most of them are related to heuristic algorithms; see, e.g., Quilliot et al. [13], Rainer-Harbach et al. [14] and Raviv et al. [15]. As mentioned before, the CVRP and the SDVRP are particular cases of the SD1PDTSP; see Archetti et al. [1] for a recent exact algorithm for the SDVRP, and Archetti and Speranza [2] for a survey on SDVRP and related problems.

This paper is organized as follows. Section 2 defines the SD1PDTSP and Section 3 gives examples of it and variants. Section 4 describes two mathematical models to solve the SD1PDTSP. Sections 5 and 6 describe methods to check if a relaxed solution is feasible and how to discard infeasible solutions, respectively. Section 7 describes a branch-and-cut algorithm to solve the SD1PDTSP. Finally, computational results are analyzed in Section 8.

## 2. Problem definition and notation

We introduce some notation. We represent by $I = \{1, 2, \ldots, n\}$ the set of locations, where 1 is the depot and $\{2, \ldots, n\}$ is the set of customers. For each customer $i$, let $p_i$ be the units of commodity in $i$ before starting the service at $i$ (i.e. the initial inventory level), and $p'_i$ be the units of commodity desired in $i$ at the end of the service (i.e. the final inventory level). Let $d_i = p'_i - p_i$ be the demand of $i$. When $d_i > 0$, the location $i$ requires a total amount of $d_i$ units of product from the network, that is, it corresponds to a delivery customer. When $d_i < 0$, the location $i$ gives a total amount of $-d_i$ units of product to the network, that is, it corresponds to a pickup customer. In addition, let $q_i$ be the storage capacity associated with customer $i$, meaning that this location can store between 0 and $q_i$ units of the commodity at a time. For consistency, we assume $0 \leq p_i \leq q_i$ and $0 \leq p'_i \leq q_i$ for each customer $i$. The capacity of the vehicle is denoted by $Q$, a-priori known. We denote by $c_{ij}$ the travel cost for the vehicle to go from $i$ to $j$, no matter its load, with $i, j \in I$ and $i \neq j$. For each $i \in I$, let $m_i$ be a given value representing the maximum number of visits allowed to location $i$.

Since the initial load of the vehicle when leaving the depot is unfixed, and it can be any value between 0 and $Q$, we do not consider any inventory constraint at the depot. Indeed, a realistic assumption allows the depot to provide or absorb whatever amount of commodity rest in the vehicle after having performed all visits. Hence, we assume $d_1 = -\sum_{i=2}^{n} d_i$, $q_1 = \infty$, and fix neither $p_1$ nor $p'_1$.

A solution for the SD1PDTSP can be represented by a sequence of visits to locations, starting and ending at the depot, i.e., $(v_0, \ldots, v_k)$ with $v_r \in I$ for $r = 0, \ldots, k$ and $v_0 = v_k = 1$. Each location $i$ must be at least once in the sequence, and at most $m_i$ times. Parameter $k$ represents the total number of the visits. In addition, a solution must be associated with the quantity picked up or delivered at each visit $v_r$.

## 3. Examples of the solutions

As observed above, there are many closely related problems to SD1PDTSP, which does not help to compare models and algorithms in the literature. This section illustrates the

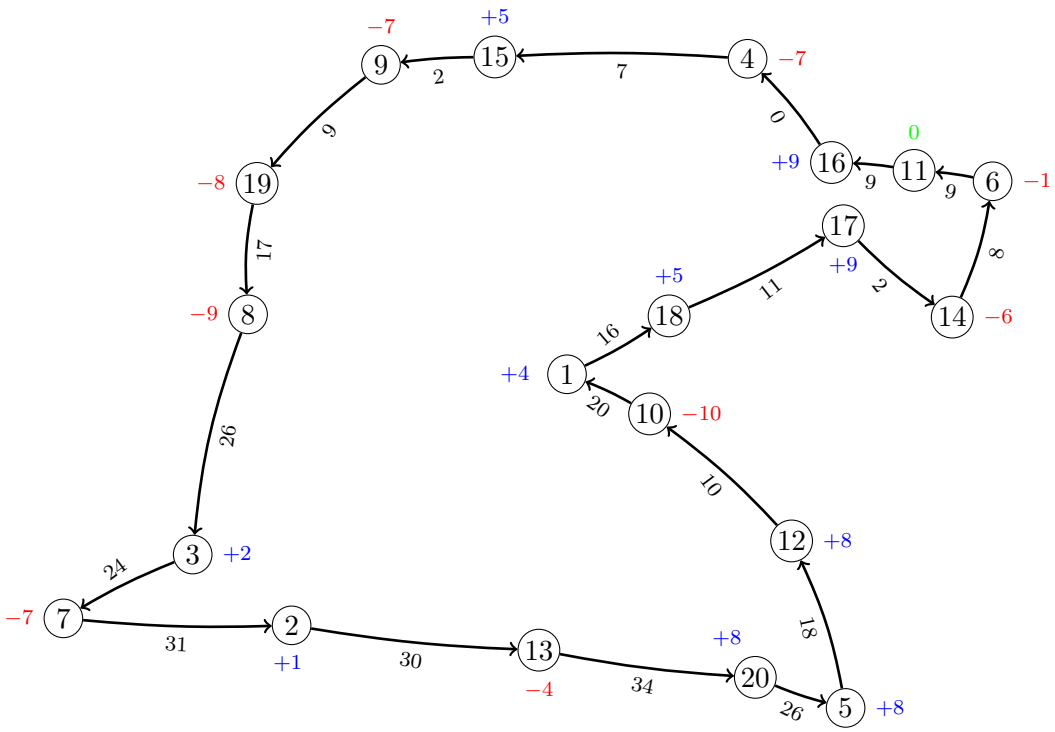


Figure 1: Instance `n20D` with $Q = 35$ and unfixed initial load in $[0, Q]$. Optimum cost 3734.

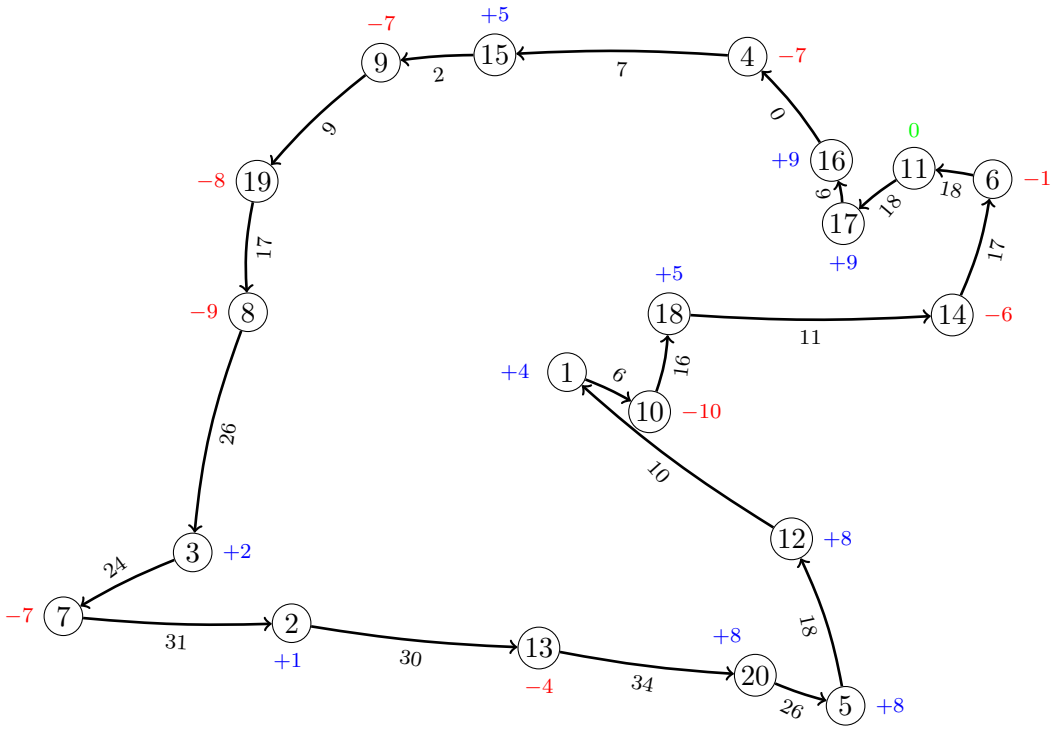


Figure 2: Instance `n20D` with $Q = 35$ and unfixed initial load in $[0, 10]$. Optimum cost 3830.

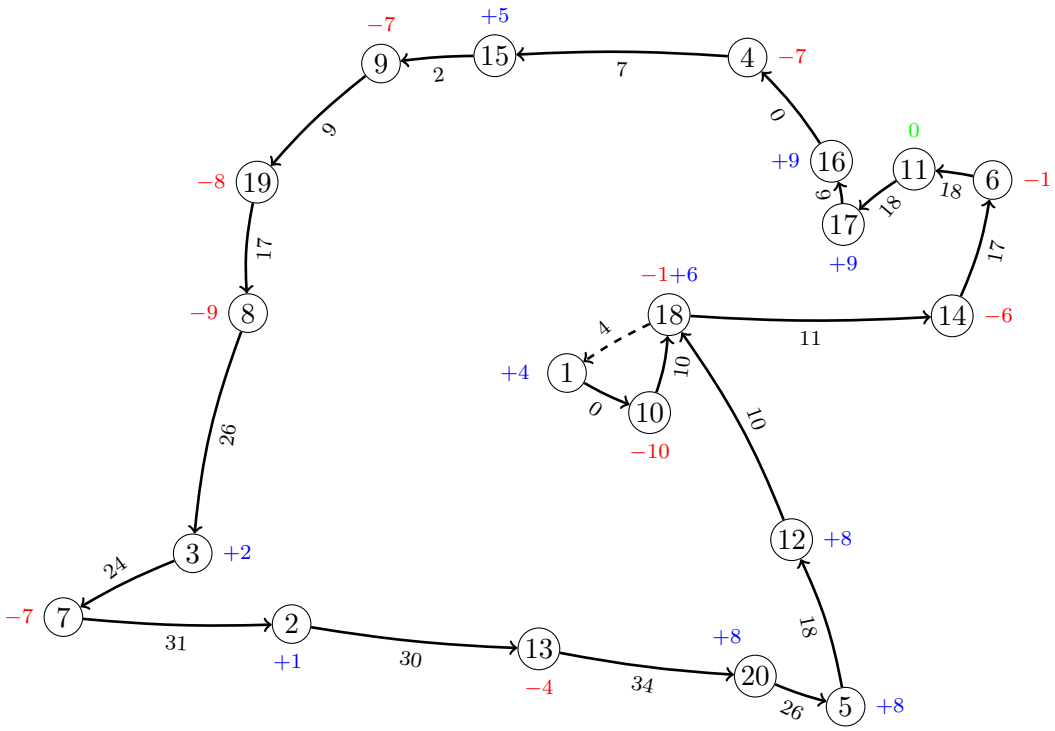


Figure 3: Instance `n20D` with $Q = 35$ and initial load fixed to 0. Optimum cost 3994.

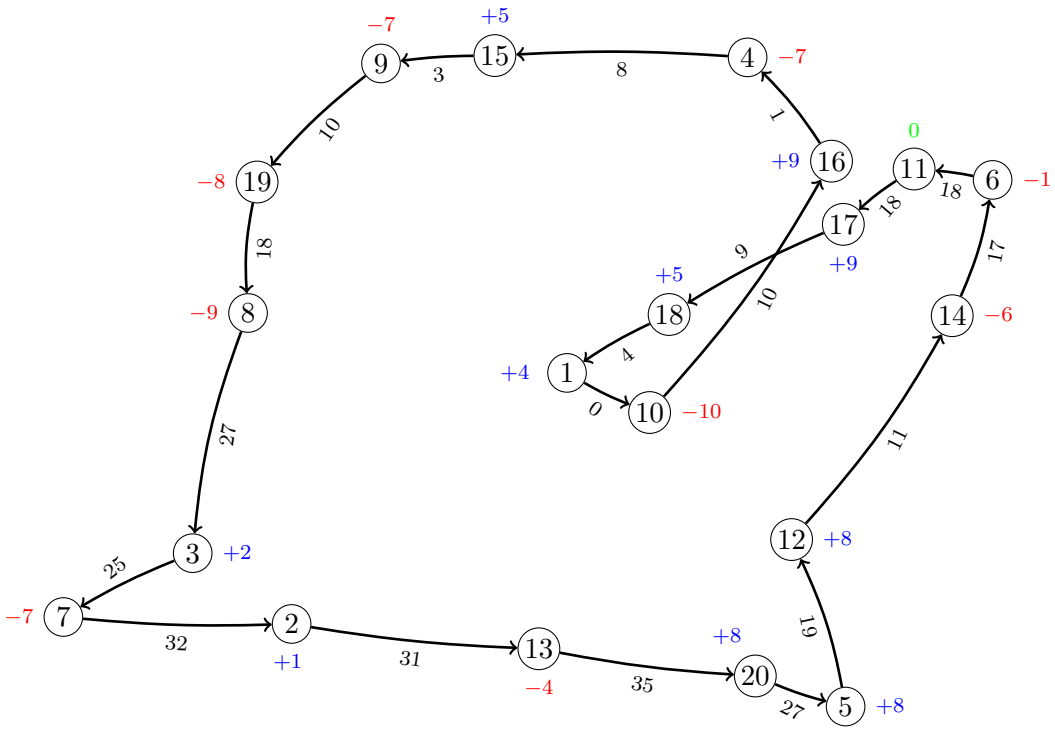


Figure 4: Instance `n20D` with $Q = 35$, initial load fixed to 0 and non-preemption. Optimum cost 4146.

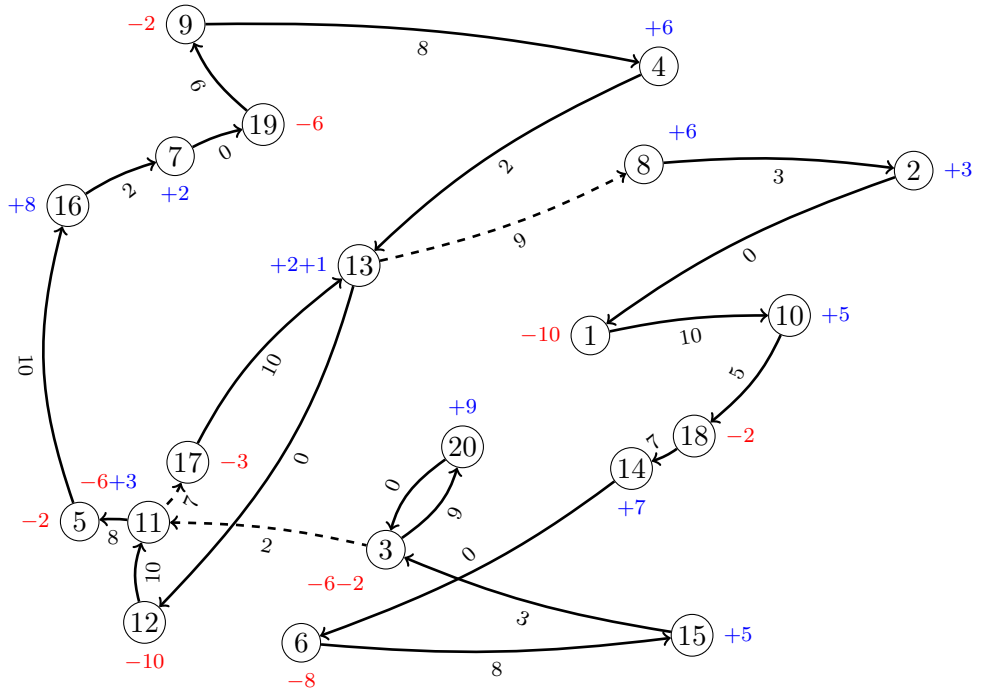


Figure 5: Instance `n20C` with $Q = 10$ and unfixed initial load in [0,10]. Optimum cost 6012.

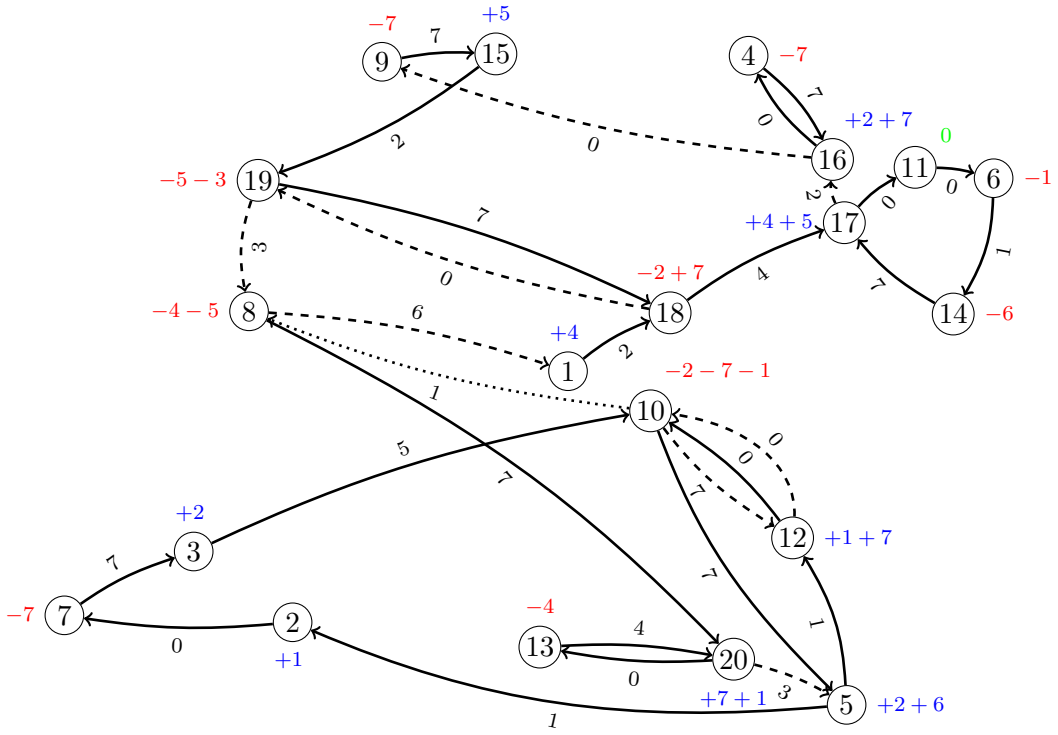


Figure 6: Instance `n20D` with $Q = 7$, unfixed initial load and $m_i = \infty$. Optimum cost 8187.

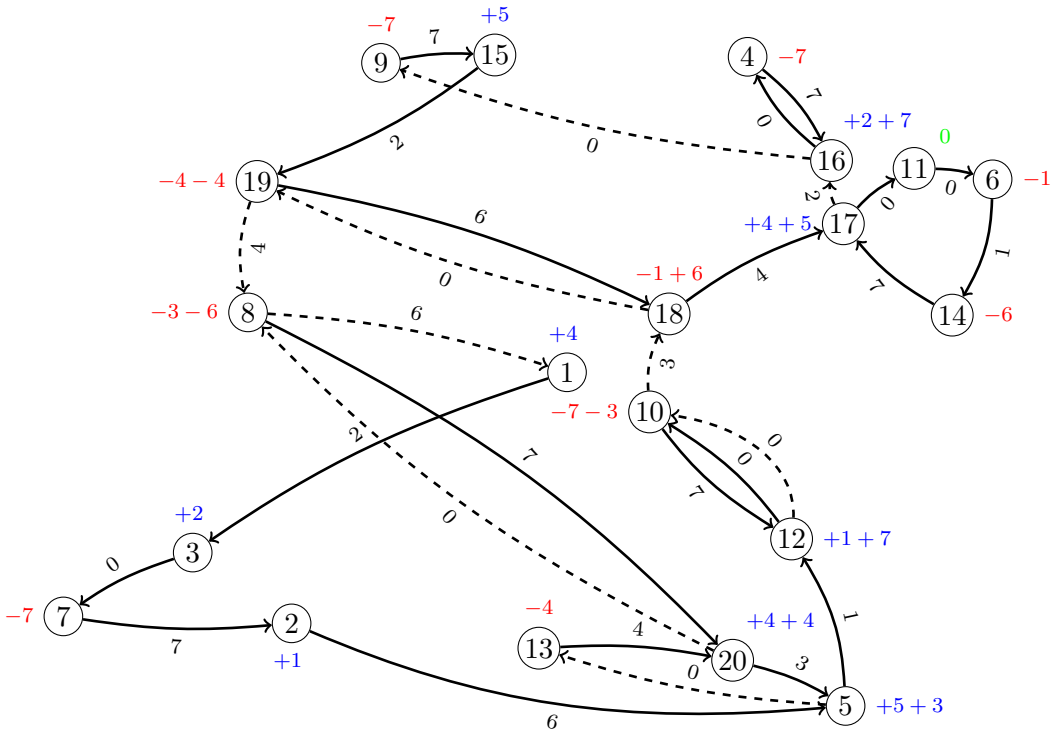


Figure 7: Instance `n20D` with $Q = 7$, unfixed initial load, $m_i = \max\{2, \lceil |d_i|/Q \rceil\}$. Optimum 8224.

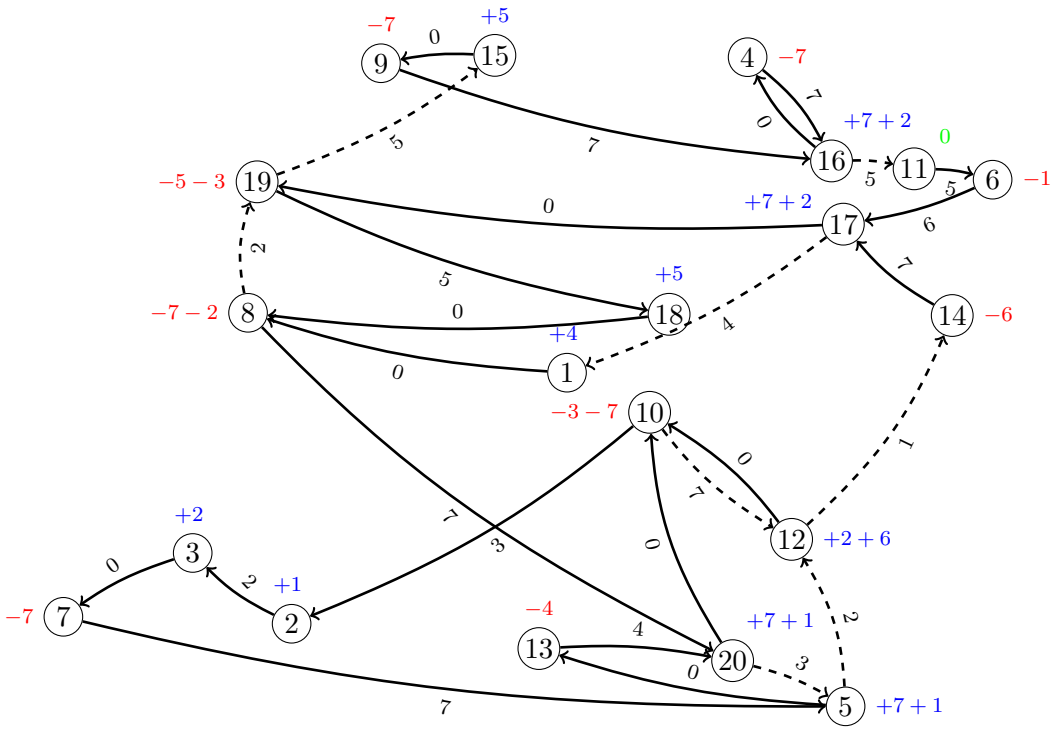


Figure 8: Instance `n20D` with $Q = 7$, unfixed initial load, $m_i = \max\{1, \lceil |d_i|/Q \rceil\}$. Optimum cost 8294.

impact of some assumptions on the optimal solution of small instances. We use the instance `n20D` used in Cruz et al. [4] and the instance `n20C` used in Erdogan et al. [7]. They have 20 locations and belong to the benchmark collection described and used in Section 8. We represent the optimal solutions of several problem variants through graphs in Figures 1–8. Each location $i \in I$ is represented by a circle in the graph. The number inside the circle is $i$ and the number close to the circle is $d_i$. When a location is visited more than once, $d_i$ is split among the quantities picked up or delivered on each visit. The number close to each arc corresponds to the load of the vehicle through this arc. To determine the route of the vehicle, the first, second and third time that the vehicle leaves a location is represented by a bold, dashed and dotted arc, respectively. Figures 1–4 are optimal solutions for problem variants with `n20D` and $Q = 35$. Figure 5 shows the solution of `n20C` with $Q = 10$ reported in Erdogan et al. [7]. Since splitting occurs rarely with large capacity values, Figures 6–8 refer to other variants with `n20D` and $Q = 7$. As in Chemla et al. [3] and Cruz et al. [4], we represent the locations with numbers from 1 to 20, whereas in Erdogan et al. [7] the locations are represented from 0 to 19; in Chemla et al. [3] and Erdogan et al. [7] the information near location $i$ is $[p_i, p'_i]$ whereas in Cruz et al. [4] it is $-d_i$.

### *3.1. Zero-demand customers*

Customer 11 in `n20D` has zero demand, but it is required to be in a SD1PDTSP route. When visiting zero-demand customers is optional (as in Chemla et al. [3], Erdogan et al. [7], Cruz et al. [4]) the optimal route may follow a different sequence of customers. The optimal SD1PDTSP of the instance `n20D` with $Q = 35$ (shown in Figure 1) follows $(1, 18, 17, 14, 6, 11, 16, \ldots)$, but without customer 11 the optimal SD1PDTSP follows $(1, 18, 14, 6, 17, 16, \ldots)$.

### *3.2. Initial load of the vehicle*

Figures 1–3 show the optimal solutions of the instance `n20D` with $Q = 35$ when the initial load of the vehicle is unfixed, upper bounded by $p_1$, and fixed to zero, respectively. We observe that the optimal costs are larger when the initial load of the vehicle is more restricted. Figure 1 is also the optimal TSP solution.

Salazar-González and Santos-Hernández [16] find optimal solutions like the one in Figure 1, which is the SD1PDTSP. Erdogan et al. [7] and Cruz et al. [4] aim at finding routes like the one in Figure 2 (relaxing the need of visiting zero-demand customers), which is the SBRP. Chemla et al. [3] look for solutions like the one in Figure 3 (relaxing the need of visiting zero-demand customers), which is the SVOCPDP. It is fundamental to be aware of this diversity in problem definitions when comparing computational results in the literature.

### *3.3. Preemption*

Figure 3 shows that customer 18 is visited twice. On the first visit the customer delivers one unit of product to the vehicle ($p_{18} \geq 1$), whereas on the second visit the customer picks up six units of product from the vehicle ($d_{18} = 5$). Thus, this solution makes use of the preemption allowance at customer 18. Figure 4 shows the optimal solution when the initial load of the vehicle is fixed to zero and preemption is not allowed. The solutions in Figures 1 and 2 visit each customer once, thus they are optimal with non-preemption as well.

Erdogan et al. [7] define the non-preemptive case differently. They allow delivery customers collecting load from the vehicle on the last visit, and pickup customers delivering load to the vehicle in the last visit. Our definition of non-preemption is the one in Cruz et al. [4] described by "not using stations as buffers". On the particular instance `n20D` with $Q = 35$ the non-preemptive solution of Erdogan et al. [7] coincides with our non-preemptive solution (except for not visiting customer 11), but this coincidence does not occur in general as illustrated in Figure 5. It is an example of route depicted in Erdogan et al. [7], corresponding to their optimal solution for the instance `n20C` with $Q = 10$. Customer 11 is a pickup customer ($d_{11} = -3$) visited twice: the vehicle collects 6 units in the first visit and delivers 3 units in the second visit. Under their problem definition this solution is optimal both for their preemptive and non-preemptive variants, whereas it is infeasible for our non-preemptive case.

### *3.4. Limiting the number of visits to a customer*

SD1PDTSP assumes a given limit $m_i$ on the number of visits to customer $i$. Note that, for an instance to be feasible, $m_i \geq \lceil |d_i|/Q \rceil$ is necessary (not sufficient). Figures 6–8 show the

optimal solutions of the instance `n20D` with $Q = 7$. The routes are optimal solutions when the initial load is unfixed and $m_i = \infty$, $m_i = \max\{2, \lceil |d_i|/Q \rceil\}$ and $m_i = \max\{1, \lceil |d_i|/Q \rceil\}$, respectively.

## 4. Mathematical Models

This section presents two MILP models for the SD1PDTSP.

### *4.1. First model*

Let $V_i$ be an ordered set of $m_i$ vertices representing potential visits to location $i \in I$. The vertices in $V_i$ are named so $i_k$ represents the $k$-th visit to $i$. With the depot represented by location 1, a feasible route for the vehicle must start and end from the vertex $1_1$. Vertices in $V_1 \setminus \{1_1\}$ are also in the route when the vehicle returns to the depot more than once. The set $V$ defined as the union of all $V_i$ contains the vertices of a directed graph $G_A = (V, A)$ where $A$ is the arc set connecting vertices associated with different locations. For a given subset $S$ of vertices, we write $\delta_A^+(S) = \{(v, w) \in A : v \in S, w \notin S\}$ and $\delta_A^-(S) = \{(v, w) \in A : v \notin S, w \in S\}$. Given an arc $a = (v, w)$ we also denote the cost $c_a$ from $v$ to $w$ as the travel cost $c_{ij}$ from the location $i$ associated with $v$ to the location $j$ associated with $w$.

Let us consider the following mathematical variables. For each arc $a \in A$, a binary variable $x_a$ assumes value 1 if and only if the route includes $a$, and a continuous variable $f_a$ represents the load of the vehicle when traversing $a$. For each vertex $v \in V$, a binary variable $y_v$ assumes value 1 if and only if the route includes $v$, and a continuous variable $g_v$ determines the number $|g_v|$ of units delivered (if $g_v > 0$) or collected (if $g_v < 0$) when performing the visit $v$. Then, the SD1PDTSP can be formulated as:

$$\min \sum_{a \in A} c_a x_a \tag{1}$$

subject to:

$$y_{i_1} = 1 \qquad \text{for all } i \in I \tag{2}$$

$$\sum_{a \in \delta_A^+(v)} x_a = \sum_{a \in \delta_A^-(v)} x_a = y_v \qquad \text{for all } v \in V \tag{3}$$

$$\sum_{a \in \delta_A^+(S)} x_a \geq y_v + y_w - 1 \qquad \text{for all } S \subseteq V \text{ , } v \in S \text{ , } w \in V \setminus S \tag{4}$$

$$\sum_{a \in \delta_A^+(S) \setminus \delta_A^+(1_1)} x_a \geq y_{i_{l+1}} \qquad \text{for all } i \in I \text{ , } l = 1, \ldots, m_i - 1$$

$$S \subseteq V : \ 1_1, i_l \in S \text{ , } i_{l+1} \in V \setminus S \tag{5}$$

$$\sum_{a \in \delta_A^-(v)} f_a - \sum_{a \in \delta_A^+(v)} f_a = g_v \qquad \text{for all } i \in I \text{ , } v \in V_i \tag{6}$$

$$0 \leq f_a \leq Q x_a \qquad \text{for all } a \in A \tag{7}$$

$$\sum_{v \in V_i} g_v = d_i \qquad \text{for all } i \in I \tag{8}$$

$$0 \leq p_i + \sum_{1 \leq k \leq l} g_{i_k} \leq q_i \qquad \text{for all } i \in I \text{ , } l = 1, \ldots, m_i - 1 \tag{9}$$

$$-q_i y_{i_l} \leq g_{i_l} \leq q_i y_{i_l} \qquad \text{for all } i \in I \text{ , } l = 2, \ldots, m_i \tag{10}$$

$$y_v, x_a \in \{0, 1\} \qquad \text{for all } v \in V \text{ , } a \in A. \tag{11}$$

Equalities (2) ensure that each customer is visited by the vehicle. Equalities (3) force the vehicle to enter and leave once in each vertex $v$ with $y_v = 1$. Inequalities (4) ensure a connected route. Inequalities (5) ensure that the vehicle visits $i_l$ before $i_{l+1}$ if $1 \leq l < m_i$ and $y_{i_{l+1}} = 1$. Note that such precedence constraints are related to a first vertex in the route, and we use $1_1$ for it. Constraints (6)–(8) ensure that the load of the vehicle is able to satisfy the demand decided at each visit. Such constraints make use of the precedence between the visits to a customer. We do not need the same requirement on the visits to the depot since SD1PDTSP assumes that the company can always provide or absorb any amount of commodity in the vehicle when entering or leaving the depot. Constraints (9) guarantee that the storage of product in a customer $i$ is always between 0 and its capacity $q_i$. Inequalities (10) impose that product can be delivered to or collected from a customer in each visit, i.e., the preemption feature.

To better understand the validity of model (1)–(11), observe that (2)–(4) and (11) ensure that $(x, y)$ represents a circuit visiting each customer. When a customer is visited several times (say, $y_{i_l} = y_{i_{l+1}} = 1$) then constraints (5) ensure that the circuit goes from $i_l$ to $i_{l+1}$ with $1_1$ not in between. In other words, starting from $1_1$, constraints (5) ensure that the circuit will continue to $i_l$, then to $i_{l+1}$, and then to $1_1$. Constraints (6)–(8) guarantee a flow of product collected or delivered within the circuit. Inequalities (9) use the fact that the visits $i_l$ and $i_{l+1}$ appear in this order on the route, and then ensure that the cumulative product at the customer $i$ is always within the limits 0 and $q_i$. Inequalities (10) guarantee that collections and deliveries are only done on visits.

This model considers the four features addressed in Section 1. However, it can easily be adapted to other variants. For example. if the customers with zero demand could be optionally visited then constraints (2) should be removed for all $i \in I$ with $d_i = 0$. If the initial load of the vehicle is desired to be fixed to (or upper bounded by) a quantity $f_0$, we must add constraint $\sum_{i \in I} f_{(1_1, i_1)} = f_0$ ( or $\leq f_0$). If preemption is forbidden at customers then constraints (10) must be replaced by:

$$g_{i_l} \geq 0 \qquad \text{for all } i \in I \setminus \{1\} : d_i \geq 0\ , \quad l = 1, \ldots, m_i$$

$$g_{i_l} \leq 0 \qquad \text{for all } i \in I \setminus \{1\} : d_i < 0\ , \quad l = 1, \ldots, m_i.$$

In such case, when preemption is not allowed in the problem, the parameters $p_i$, $p'_i$ and $q_i$ are useless since parameters $d_i$ are enough for validating a non-preemptive route, and equations (9) can be removed from the model.

As said before, Erdogan et al. [7] consider a different definition of non-preemption. They consider that, if $i$ is a delivery customer (i.e., $d_i > 0$) and $i$ is visited $z_i^*$ times with $z_i^* > 1$, the vehicle has to deliver in visits from 1 to $z_i^* - 1$ but the vehicle could pickup in the last visit to customer $i$. Analogously, if $i$ is a pickup customer (i.e., $d_i < 0$) and $i$ is visited $z_i^*$ times (with $z_i^* > 1$), the vehicle has to pickup in visits from 1 to $z_i^* - 1$, and the vehicle could deliver in the last visit to customer $i$. For that reason, Figure 5 is a non-preemptive route in Erdogan et al. [7] (the vehicle pickups 6 units in a first visit to customer 11 and delivers 3 units in the last visit to this customer with $d_{11} = -3$) and infeasible for our non-preemptive

SD1PDTSP. We prefer our definition (which coincides with the one in Rainer-Harbach et al. [14], Cruz et al. [4] and Quilliot et al. [13]) because it is simpler to explain and model. Moreover, our definition is more restrictive and in general the instances are more difficult to solve with our definition than with the definition in Erdogan et al. [7].

*4.2. Second model*

We present a relaxed (invalid) formulation based on a smaller graph. Let $G_B = (I, B)$ be the oriented graph where each vertex represents a location (instead of a potential visit) and each arc $b \in B$ represents an arc between two locations. Given two subsets $S, T \subset I$, we write $B(S:T) = \{(i,j) \in B : i \in S,\ j \in T\}$, $\delta_B^+(S) = B(S : I \setminus S)$ and $\delta_B^-(S) = B(I \setminus S : S)$. For each $b = (i,j) \in B$ we also write $c_b$ as the travel cost $c_{ij}$ from $i$ to $j$. Then we define the following decision variables. For each $i \in I$, let $z_i$ be an integer variable representing the number of visits to location $i$. For each arc $b \in B$, let $x_b$ be an integer variable representing the number of times the arc $b$ is traversed by the vehicle. Observe that we are using the notation $x$ for variables in both models; however, this should not confuse the reader as the subindex identifies the model univocally. For example, $x_a$ is a binary variable and $x_b$ is a general integer variable that can assume a value greater than one. Consider now the following mathematical formulation:

$$\min \sum_{b \in B} c_b x_b \tag{12}$$

subject to:

$$\sum_{b \in \delta_B^-(i)} x_b = \sum_{b \in \delta_B^+(i)} x_b = z_i \qquad \text{for all } i \in I \tag{13}$$

$$1 \le z_i \le m_i \qquad \text{for all } i \in I \tag{14}$$

$$\sum_{b \in \delta_B^+(S)} x_b \ge \max \left\{ 1, \left\lceil \frac{|\sum_{i \in S} d_i|}{Q} \right\rceil \right\} \qquad \text{for all } S \subset I \tag{15}$$

$$x_b \ge 0 \text{ and } x_b \in \mathbb{Z} \qquad \text{for all } b \in B. \tag{16}$$

Equations (13) forces the same number of arcs in the route going in and out for each location. Inequations (14) limit the number of visits to each location. Inequations (15) ensure the connectivity of the route and the vehicle capacity.

Clearly, all SD1PDTSP solutions satisfy the above formulation, but there may be solutions of the formulation which do not correspond to SD1PDTSP solutions: note that the system (13)–(16) considers $d_i$ but neither $p_i$ nor $q_i$, thus any disaggregation of a solution of this system into a route in $G$ could violate the constrains (8)–(10). Therefore, to solve the SD1PDTSP with model (12)–(16), we need a procedure to determine whether an integer solution of (13)–(16) is feasible for the SD1PDTSP, and a procedure to discard infeasible solutions. The former is tackled in Section 5, whereas the latter is described in Section 6.

## 5. Checking the feasibility of an integer solution of the relaxed model

Given an integer solution $(z^*, x^*)$ of model (12)–(16), this section describes five methods to check if $(z^*, x^*)$ yields a feasible solution for the SD1PDTSP. One of them consists of solving a MIP model extracted from (1)–(11). Other three methods are based on enumerating the Eulerian circuits that can be built from $(z^*, x^*)$ and checking if each circuit is feasible for the SD1PDTSP. We use the term *Eulerian circuit* from an integer solution $(z^*, x^*)$ to denote a sequence of arcs leading to a feasible SD1PDTSP route which uses $x^*_a$ times each arc $a \in A$, and therefore visits $z^*_i$ each location $i \in I$. The last method checks the feasibility of partial circuits along a first-depth search.

### *5.1. Solving a MIP*

A first method considers a linear system from the model (1)–(11) while assuming that customer $i$ is visited $z^*_i$ times. Let $G^*_A = (V^*, A^*)$ be the subgraph of $G_A$ induced by $V^* := \cup_{i \in I}\{i_1, \ldots, i_{z^*_i}\}$. For each arc $a$ in $A^*$, let us consider a binary variable $x_a$ assuming value 1 if and only if $x^*_a > 0$, and a continuous variable $f_a$ representing the vehicle load when traversing $a$. For each vertex $v$ in $V^*$, let us consider a continuous variable $g_v$ representing the amount delivered (if $g_v > 0$) or collected (if $g_v < 0$) when performing the visit $v$. The new linear system contains (2)–(11) with $V$ replaced by $V^*$ and the $y_v$ variables fixed to value 1 for all $v \in V^*$. The variables in the system are now $x_a$, $f_a$ and $g_v$, and the constraints

are:

$$\sum_{a\in\delta^+_{A^*}(v)} x_a = \sum_{a\in\delta^-_{A^*}(v)} x_a = 1 \qquad \text{for all } v \in V^* \tag{17}$$

$$\sum_{a\in\delta^+_{A^*}(S)} x_a \geq 1 \qquad \text{for all } S \subseteq V^* \tag{18}$$

$$\sum_{a\in\delta^+_{A^*}(S)\setminus\delta^+_{A^*}(1_1)} x_a \geq 1 \qquad \text{for all } i \in I\ ,\ l = 1,\ldots,z^*_i - 1\ (i_l \neq 1_1),$$
$$S \subseteq V^* :\ 1_1, i_l \in S\ ,\ i_{l+1} \in V^* \setminus S \tag{19}$$

$$\sum_{a\in\delta^-_{A^*}(v)} f_a - \sum_{a\in\delta^+_{A^*}(v)} f_a = g_v \qquad \text{for all } v \in V^* \tag{20}$$

$$0 \leq f_a \leq Q x_a \qquad \text{for all } a \in A^* \tag{21}$$

$$\sum_{1\leq l\leq z^*_i} g_{i_l} = d_i \qquad \text{for all } i \in I \tag{22}$$

$$0 \leq p_i + \sum_{1\leq k\leq l} g_{i_k} \leq q_i \qquad \text{for all } i \in I\ ,\ l = 1,\ldots,z^*_i - 1 \tag{23}$$

$$-q_i \leq g_{i_l} \leq q_i \qquad \text{for all } i \in I\ ,\ l = 2,\ldots,z^*_i \tag{24}$$

$$x_a \in \{0,1\} \qquad \text{for all } a \in A^* \tag{25}$$

$$\sum_{a=(i_k,j_l):(i,j)=b} x_a = x^*_b \qquad \text{for all } b \in B. \tag{26}$$

The explanation of each constraint can be deduced from the explanation of (2)–(11). Although the numbers of constraints in (18) and (19) are exponential, they can be managed dynamically by solving max-flow problems. A solution of the new linear system (17)–(26) can be seen as a certificate that the solution $(z^*, x^*)$ from (12)–(16) represents a valid route for the SD1PDTSP.

*5.2. Enumerating all circuits*

The difficulty of checking the SD1PDTSP feasibility of a solution is due to the potential existence of many Eulerian circuits where a location $i$ is visited more than once, with no one accepting partial collections or deliveries to certificate the validity of the solution. Indeed, starting from the depot and following $x^*$, one must decide how to leave from a location $i$ with $z^*_i \geq 2$. It is not only to decide which will be the next customer to visit, but also which was

the partial demand served to $i$. Erdogan et al. [7] suggest enumerating all Eulerian circuits in $(z^*, x^*)$ using each arc $b$ exactly $x_b^*$ times in a first step, and then finding a procedure to check whether there exist loads for the vehicle on each Eulerian circuit. The enumeration quits when it finds the first Eulerian circuit with loads certifying the feasibility of the solution. We now summarize both steps.

To enumerate all Eulerian circuits in a given a solution $(z^*, x^*)$ one can perform a depth-first search in the supporting graph of $x^*$, i.e., the graph $G_B^* = (I, B^*)$ where $B^* = \{b \in B : x_b^* > 0\}$. Note that $x_b^*$ can be greater than one, thus arc $b$ must be used more than once in the depth-first search (i.e., $G_{B^*}$ is a multigraph). The search can be pruned if a vertex in the graph $G^*$ removing all used arcs is unreachable. When the vertex degrees in $G_B^*$ are large, the task of listing all possible Eulerian circuits may be quite time consuming.

Chemla et al. [3] solve a max-flow problem to check the feasibility of each circuit. Rainer-Harbach et al. [14] adapt the procedure for the non-preemptive case and many vehicles. Similarity, Erdogan et al. [7] solve a max-flow problem using a linear programming based on a time extended network. When preemption is forbidden, they propose another linear programming over the time extended network to check the feasibility of a circuit.

Appendix A describes a new procedure to check the feasibility of a circuit. It is based on limiting (with intervals) the amount of product delivered or collected at a location on each visit, and limiting the load of the vehicle when leaving a location on each visit. This procedure can be applied to each circuit of a given solution $(z^*, x^*)$ as the procedures described by Chemla et al. [3], Rainer-Harbach et al. [14] and Erdogan et al. [7]. In addition, the new procedure can also be applied on partial routes of a circuit while enumeration all of them. This integrated use of the new procedure has the advantage that an invalid partial circuit will not be evaluated several times, and the procedure helps to prune the depth-first search.

Summarizing, there are five methods to check the feasibility of a solution:

MIP: solving the MIP model in Section 5.1,

LP: solving the linear program in Erdogan et al. [7],

MF: solving the max-flow problem in Chemla et al. [3] and Rainer-Harbach et al. [14],

interval: applying the procedure in Appendix A on each Eulerian circuit,

integrated: applying the procedure in Appendix A on partial routes.

The next section shows computational experiments using each one.

*5.3. Computational results of the five methods to check the feasibility*

To select the best method among the five, this section analyzes some computational experiments. For each instance, the relaxed model (12)–(16) was solved and its solution was used as input data for each method. Table 1 shows the computational results of the five methods on 1860 instances described in Section 8, grouped in five families of instances, each one identified here with the number of the table where it is detailed in Section 8. In addition, Table 1 also shows the results of three subfamilies of the instances in Tables 4 and 6 to better understand the aggregated results of these two families. The column heading "Ins." refers to the number of instances of each family. The column heading "$\bar{z_i^*}$" displays the average number of visits to each location in each family of instances (i.e. $\bar{z_i^*} = \sum_{i \in I} z_i^*/n$). The column heading "Feas." shows the percentage of the number of instances in each family where the relaxed solution was feasible for the SD1PDTSP. The column heading "Circuits" refers to the average number of Eulerian circuits in the relaxed master solution $(z^*, x^*)$. It is not necessary to enumerate and analyze all the Eulerian circuits that can be generated from $(z^*, x^*)$, unless all are invalid. We have enumerated all of them outside these algorithms just to count them and show the number in this table. The column heading "Explored" refers to the average number of Eulerian circuits analysed from the relaxed solution by the methods *LP*, *MF* and *interval*. The following five columns refer to the average computational time (in seconds) consumed by each method.

From Table 1, the instances requiring more feasibility validations correspond to the ones in Table 6, which are instances where preemption is forbidden. In particular the checking operation is heavier on the instances where the vehicle capacity is smaller compared to the customer demands. On those instances, the fastest method is *integrated*.

| Table | Ins. | $\bar{z}_i^*$ | Feas. | Circuits | Explored | *MIP* | *LP* | *MF* | *interval* | *integrated* |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 60 | 1.22 | 95.0 | 8.1 | 1.02 | 0.059 | 0.000249 | 0.000020 | 0.000011 | 0.000027 |
| 3 | 450 | 0.97 | 41.8 | 1.1 | 1.02 | 0.155 | 0.000218 | 0.000020 | 0.000005 | 0.000016 |
| 4 | 450 | 1.21 | 83.1 | $1.9 \times 10^6$ | 1.23 | 0.575 | 0.000301 | 0.000032 | 0.000040 | 0.000062 |
| 5 | 450 | 0.93 | 14.2 | 1.1 | 1.05 | 0.096 | 0.000146 | 0.000018 | 0.000003 | 0.000011 |
| 6 | 450 | 1.17 | 38.9 | $3.5 \times 10^6$ | 9626.33 | 0.453 | 1.411707 | 0.218721 | 0.140280 | 0.036523 |
| 4 ($Q = 10$) | 50 | 1.97 | 100.0 | $17.2 \times 10^6$ | 1.00 | 3.356 | 0.000428 | 0.000056 | 0.000153 | 0.000198 |
| 4 ($15 \leq Q \leq 25$) | 150 | 1.31 | 98.7 | 48.8 | 1.47 | 0.301 | 0.000355 | 0.000041 | 0.000052 | 0.000080 |
| 4 ($Q \geq 30$) | 250 | 1.00 | 70.4 | 1.5 | 1.14 | 0.184 | 0.000244 | 0.000022 | 0.000010 | 0.000025 |
| 6 ($Q = 10$) | 50 | 1.91 | 82.0 | $31.4 \times 10^6$ | 86543.78 | 2.950 | 12.696281 | 1.966221 | 1.261783 | 0.328368 |
| 6 ($15 \leq Q \leq 25$) | 150 | 1.27 | 57.3 | 38.4 | 28.79 | 0.191 | 0.002738 | 0.000715 | 0.000237 | 0.000090 |
| 6 ($Q \geq 30$) | 250 | 0.96 | 19.2 | 1.4 | 1.37 | 0.110 | 0.000173 | 0.000025 | 0.000006 | 0.000013 |

Table 1: Average time when checking feasibility.

## 6. Discarding an integer solution of the relaxed model

If a method concludes that $(z^*, x^*)$ is not feasible for the SD1PDTSP, this relaxed solution must be discarded in the master model (13)–(16). For this purpose Erdogan et al. [7] include new variables $w_{bk}$ representing the integer numbers $x_b^*$ in binary format, i.e.

$$x_b = \sum_{k=0}^{\lfloor \log_2 m_b \rfloor} 2^k w_{bk},$$

where $m_b = 2\min\{m_i, m_j\}$ is an upper limit of the variable $x_b$ for each arc $b = (i, j) \in B$. Then the inequality to cut-off the infeasible solution $(z^*, x^*)$ is

$$\sum_{b \in B, k \in \{0,\ldots,\log_2 m_b\}: w_{bk}^* = 0} w_{bk} + \sum_{b \in B, k \in \{0,\ldots,\log_2 m_b\}: w_{bk}^* = 1} (1 - w_{bk}) \geq 1. \tag{27}$$

We propose the use of a stronger inequality taking advantage of the method *integrated* which returns the subset $S$ defining the locations reachable from the depot. Indeed, given $S$, any path with all arcs inside or leaving $S$ cannot be part of a feasible SD1PDTSP solution, thus

$$\sum_{\substack{b \in B(S) \cup \delta_B^+(S), \\ k \in \{0,\ldots,\log_2 m_b\}: w_{bk}^* = 0}} w_{bk} + \sum_{\substack{b \in B(S) \cup \delta_B^+(S), \\ k \in \{0,\ldots,\log_2 m_b\}: w_{bk}^* = 1}} (1 - w_{bk}) \geq 1. \tag{28}$$

Figure 1 shows the optimal solution of the model (12)–(16) on the instance `n20D` with $Q = 35$. If the initial load of the vehicle is unfixed, but required to be not larger than 10,

then this solution is infeasible for the SD1PDTSP. The method *integrated* returns the subset $S = \{1, 18\}$ of locations which are reachable from the depot, and $S$ yields an inequality (28) violated which is stronger (smaller left-hand side) than inequality (27). Note that $\{b \in B(S) \cup \delta_B^+(S) : x_b^* > 0\} = \{(1, 18), (18, 17)\}$ for this solution.

## 7. Branch-and-cut algorithm

The above sections suggest a SD1PDTSP algorithm which consists of solving a master model including (12)–(16) and the constraints (15) separated from the solution $(z^*, x^*)$ by three heuristic procedures as follows.

The first procedure builds a list of candidates $S$. It uses two integer parameters $P_1$ and $P_2$ for this construction. The procedure starts finding the $P_1$ subsets of cardinality two $\{i, j\}$ maximizing $x^*_{(i,j)} + x^*_{(j,i)}$. After, each set of cardinality $k$ (with $3 \leq k \leq P_2$) is generated from a set of cardinality of size $k - 1$ in the list, inserting a vertex $j$ such that $j \in I \setminus S$, $\sum_{i \in S}(x^*_{(i,j)} + x^*_{(j,i)}) > 0$ and $S \cup \{j\}$ is not in the list. If there are several vertices $j$ verifying these conditions, the one with maximum $\sum_{i \in S}(x^*_{(i,j)} + x^*_{(j,i)})$ is preferred. Parameters $P_1$ and $P_2$ are set to $n$ and $n/2$, respectively, in our implementation.

The second procedure computes max-flow problems to guarantee $\sum_{b \in \delta_B^+(S)} x_b \geq 1$, which is folklore in the TSP literature. Following a separation procedure described in [9], the third procedure guarantees $\sum_{b \in \delta_B^+(S)} x_b \geq |\sum_{i \in S} d_i|/Q$ by also solving max-flow problems.

Fractional solutions are discarded through a classical binary branching procedure with priorities on the $z$ variables.

We use the method *integrated* described in Section 5 to check whether $(z^*, x^*)$ corresponds to a feasible SD1PDTSP solution. If it is infeasible, the algorithm introduces a constraint (28); otherwise, the solution is classified as SD1PDTSP feasible, and the algorithm continues.

We now detail other elements considered in our branch-and-cut algorithm.

### *7.1. Cuts from infeasible linear system*

Given a solution $(z^*, x^*)$ where $z^*$ is integer (i.e., $z_i^* \in \mathbb{Z}$ for all $i \in I$), one could solve the dual problem of (17)–(26) relaxing the integer constraints (25). If this linear system is infeasible, the dual cone is unbounded, thus there is a ray defining a violated constraint

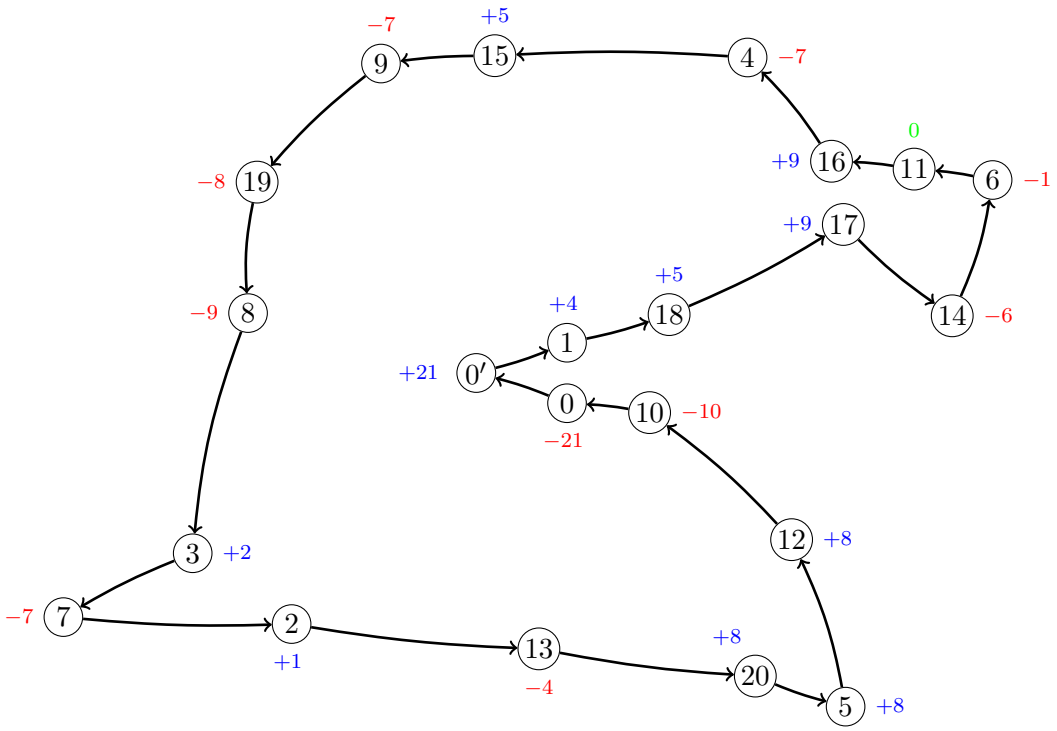


Figure 9: Infeasible solution of instance `n20D` with $Q = 35$ and unfixed initial load in [0,10].

to include in the master model (12)–(16). This is the classical Benders' decomposition approach. We made computational experiments with this approach and, unfortunately, solving the dual relaxed problem was rarely infeasible while it was time consuming. Thus, we did not active it in our implementation for the final experiments.

*7.2. Transformation of the relaxed model*

As said before, our formulation can be adapted to deal with the problem variant where the initial load of the vehicle is restricted (bounded by or fixed to a quantity). This section shows the adaptation when the initial load is required to be bounded by a value $p_0$, smaller than $Q$. We transform this variant to the SD1PDTSP as follows.

We add two dummy customers 0 and $0'$ with the following parameter values: $d_0 = p'_0 = p_1 + d_1 - Q$, $d_{0'} = p'_{0'} = -d_0$, $p_0 = Q$, $p_{0'} = 0$ and $q_0 = q_{0'} = Q$. Travel costs are defined by $c_{00'} = c_{0'1} = 0$ and $c_{i0} = c_{i1}$ for each $i \in I$. In addition, we force customers 0 and $0'$ to be visited only once, the arcs $(0, 0')$ and $(0', 1)$ to be in the route, and the arc $(1, 0)$ to do not be in the route.

Observe that with this transformation the load in the vehicle through arc $(0, 0')$ must be in the interval $[-p_1 - d_1 + Q, Q]$, the load through arc $(0', 1)$ must be in the interval $[0, p_1 + d_1]$ and, if customer 1 is visited only once, the vehicle must leave it with load in the interval $[0, p_1]$. More precisely, let $S'$ be a subset of customers such that $d(S') > p_1$; then the subset $S = S' \cup \{0', 1\}$ defines an inequality (15) with right-hand side greater than one. Figure 9 shows the relaxed solution associated to the solution in Figure 1 for the transformed instance considering the initial load of the vehicle bounded by $p_1 = 10$. The relaxed problem (12)–(16) is infeasible because constraint (15) is violated for the subset $S = \{0', 1, 18, 17\}$.

When preemption is forbidden and the initial load of the vehicle is bounded by $\max\{0, -d_1\}$, then we apply a similar transformation but replacing $p_1$ by $\max\{0, -d_1\}$.

*7.3. Temporarily discarding a relaxed solution*

We have observed that checking if a relaxed solution $(x^*, z^*)$ is SD1PDTSP feasible may consume quite long time when the total number of visits is large with respect to the minimum number of possible visits. More precisely, let $m'_i = \lceil |d_i|/Q \rceil$ the minimum number of visits to locations $i$; if the value $\sum_{i \in I}(z^*_i - m'_i)$ is large, even the *integrated* method may have bad performances.

To do not spend much time checking the feasibility of relaxed solutions which their costs are greater than the optimal value, we set a time limit (5 seconds) for this task. If the time limit is reached, then this solution (and its cost) is stored in a list (ordered by its cost). While upper and lower bounds of the branch-and-cut algorithm are being updated, this list is updated as well. If the list contains a solution with cost greater or equal to the current upper bound, this solution is removed from the list. If the list contains a relaxed solution with cost smaller than the current lower bound, the procedure to check the feasibility of the solution is executed without time limit. This approach showed better performances on our experiments and hardly ever the feasibility-checking procedure was executed twice.

*7.4. Undirected master problem formulation for symmetric instances*

The relaxed model (12)–(16) allows asymmetric travel costs. Although it can solve instances with symmetric travel costs, we have also implemented an alternative version of

the model for these instances to reduce the number of variables in the model and speed up the execution of the branch-and-cut algorithm. For the sake of brevity, we do not write this alternative version of model (12)–(16). It has a general integer variable $x_{[i,j]}$ for each pair of location $\{i, j\}$ (edge of an undirected graph) instead of the general integer variable $x_{(i,j)}$ for each arc $(i, j)$ of the directed graph $G_B$, and it has similar constraints than (13)–(16). Note that checking if a relaxed solution is SD1PDTSP feasible can be harder with a symmetric model than with an asymmetric model because the depth-first search must evaluate the edges in both directions when the Eulerian circuits are enumerated. Nevertheless, as illustrated in the next section, we observed that it is faster to solve the symmetric relaxed model than the asymmetric relaxed model when the travel costs are symmetric.

## 8. Computational results

The branch-and-cut algorithm described in the previous sections has been executed on a computer with Intel Core i7-8700 3.20 Ghz, running Windows 10 64-bit. The code was written in C++ using the branch-and-cut framework in IBM ILOG CPLEX 12.10.

The computational results are compared with the results in Salazar-González and Santos-Hernández [16] and Erdogan et al. [7]. The algorithm described in [16] was executed on a computer with Intel Core 2 Duo CPU E8600 3.33 Ghz and IBM ILOG CPLEX 12.5. The algorithm described in [7] was executed on an IRIDIS 4 computing cluster 2.6 Ghz with IBM ILOG CPLEX 12.5. Based in the single-thread rating values[1], our computer is 1.92 times faster than the computer used in Salazar-González and Santos-Hernández [16], and 1.20 faster than the computer used in Erdogan et al. [7]. The three algorithms were executed using a single thread of the CPU and a time limit of 2 hours.

The SD1PDTSP instances in Salazar-González and Santos-Hernández [16] and Erdogan et al. [7] are based on the benchmark 1PDTSP instances proposed in Hernández-Pérez and Salazar-González [10]. These instances are publicly available[2] and generated in the following way. The customers $2, \ldots, n$ are randomly located in the square $[-500, 500]$ and have integer

[1] `https://www.cpubenchmark.net/singleThread.html`
[2] `http://hhperez.webs.ull.es/PDsite/`

demands $\tilde{d}_i$ randomly chosen in the interval $[-10, 10]$. Customer 1 is located in the central point $(0, 0)$ with a demand value such that the sum of all customer demands is zero.

The instances in Salazar-González and Santos-Hernández [16] have $d_i = \tilde{d}_i$, $p_i = 10 - d_i$, $p'_i = 10$ and $q_i = 20$ for each customer $i$, the initial load of the vehicle (when leaving customer 1) is unfixed, and customers with zero demand must be visited. The travel costs are Euclidean distances rounded to the closest integer numbers, and zero-demand customers must be visited.

The instances in Erdogan et al. [7] have $d_i = \alpha \cdot \tilde{d}_i$, $p_i = 10 \cdot \alpha$, $p'_i = \alpha \cdot (10 + \tilde{d}_i)$ and $q_i = 20 \cdot \alpha$ for each customer $i$ and $\alpha \in \{1, 3\}$. Note that the effect of multiplying the customer demands by $\alpha$ is similar to divide the vehicle capacity by $\alpha$. However, to avoid discrepancies with integer rounding operations, we decided to keep exactly their settings in Erdogan et al. [7] when comparing with their results. The travel costs are Euclidean distances truncated to integer numbers, and zero-demand customers are optionally visited. These instances were introduced by Chemla et al. [3] where a depot 0 is added as a copy of the customer 1, i.e. the instances have two locations 0 and 1 at the central point $(0, 0)$. The vehicle leaves the depot 0 with empty load, so it leaves customer 1 with at most $p_1$ units of the product.

We now discuss the computational results obtained solving the instances in Salazar-González and Santos-Hernández [16] and Erdogan et al. [7]. Table 2 displays the performance on the (preemption) instances in [16], and Tables 3–6 report on the instances in [7] with $\alpha = \{1, 3\}$ and with and without preemption. Each table shows the results taken from the related article, and the results of our two implementations (one based on directed-arc general-integer variables $x_{(i,j)}$ and the other based on undirected-arc general-integer variables $x_{[i,j]}$ as described in Section 7.4). For each table, each row shows aggregated results of ten instances. The desegregated results are publicly available[2]. Column "Sol." shows the number of instances solved to optimality over 10. Column "Time" is in CPU seconds. Column "$z_i^*$" shows the number of customers for possible values in $2, \ldots, m_i$. Columns under "check" refer to the feasibility checking procedure (the integrated method). In particular "Fea." shows the average number of solutions checked and a feasible Eulerian

circuit was found; "Inf." shows the average number of solutions checked and no feasible Eulerian circuit was found; and "Time" is the average computational time consumed by the checking method.

From Table 2, our branch-and-cut algorithms outperform the one in Salazar-González and Santos-Hernández [16] even with arc-directed variables in the master problem on these instances with symmetrical travel costs. It is interesting to observe that the first master solution was SD1PDTSP feasible in most of the instances, which is the reason for the large reduction of the computational time. It is also interesting the further speed up when using the undirected variables instead of directed ones in the master problem. As also observed in Salazar-González and Santos-Hernández [16], very few customers are visited three times by the optimal routes even if $m_i = 3$.

Tables 3 and 4 show results of our branch-and-cut algorithm on the preemptive instances in Erdogan et al. [7] with $\alpha = 1$ and $\alpha = 3$, respectively. The most relevant observation is that the new algorithm solves almost all the instances with smaller computational time. Indeed, all instances are solved with the symmetric version of the new algorithm. As once can expect, the instances need more computational effort when $n$ increases and $Q$ decreases, and in coherence the instances with $\alpha = 3$ are harder than the instances with $\alpha = 1$. On difficult instances, there is a clear advantage of solving the master model with undirected variables rather than with directed variables. In the worst case, up to an average of no more than 20 master solutions needed to be checked (for feasibility), and in general very few of these solutions were infeasible (implying that the number of inequalities (28) generated to solve these instances is very small).

In few instances, the optimal cost from our algorithm is one unit smaller that the optimal cost reported in Erdogan et al. [7]. This unexpected observation is also reported in Cruz et al. [4]: "we adopted their same convention of rounding down the values of the cost matrix to the nearest integer (floor), although we noticed that this can cause slight violations of the triangle inequality." They report average results and refer to a report[3] where the cost of the

[3] `https://arxiv.org/pdf/1605.00702v2.pdf`

best found solution for each instance is given, emphasizing the instances "in which the cost of our best feasible solution is 1 unit smaller than the cost of the optimal solution reported in Erdogan et al. [7] (possibly due to rounding issues)." Our results are coherent with the cost values in this report. Another potential reason for the one-unit difference could also be due to the relative MIP gap tolerance of IBM ILOG CPLEX (parameter `CPX_PARAM_EPGAP`) which by default is 0.0001, allowing the MIP solver to stop with a difference greater than 1 unit between the lower and upper bounds if the optimal cost is greater than 10000. We used 0.000001 for the relative MIP gap tolerance.

Tables 5 and 6 compare our algorithms with the one in Erdogan et al. [7] for the non-preemptive case, again with $\alpha = 1$ and $\alpha = 3$, respectively. To compare the optimal costs respect to the preemptive case, the tables also show Gap', computed as the percentage difference divided by the preemptive cost. Remember that we use a different definition of non-preemptive and therefore our optimal solutions may slightly differ. Our definition is more restrictive, which is confirmed by the larger value of Gap' from our approach (also from Cruz et al. [4] for $n = 20$) when compared to the value of Gap' from [7]. For example, for the instances with $n = 20$ and $Q = 35$, the gap over the preemptive case is 1.14% in [7] and 1.70% in our results.

The time consumed by the checking procedure is quite small for most of the instances. The exceptions are in Table 6, where for example the instances with $n = 60$ and $Q = 10$ are solved on average in 293.3 seconds, using an average of 186.5 seconds to check the SD1PDTSP feasibility of the master solutions (on average 11.3 solutions are feasible and 26.0 are infeasible). Recall that an infeasible master solution means that no Eulerian circuit admits vehicle loads to serve the customer demands with the capacity limitations. The values in Column $z_i^* \geq 4$ are the largest on these instances, thus the number of Eulerian circuits in a master solution was largest as well. This claim is confirmed in Table 1, showing millions of Eulerian circuits in a master solution when $\alpha = 3$ and $Q = 10$. In addition, the non-preemptive variant is more restricted and therefore it is more likely for the checking method to conclude infeasibility. Specially on these difficult instances the use of the new *integrated* method helps, as it allows confirming the infeasibility of a solution without the explicit

| | | [16] | | directed | undirected | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | $z_i^*$ | | check | | | |
| $n$ | $Q$ | Sol. | Time | Time | =2 | =3 | Fea. | Inf. | Time | Time |
| 30 | 5 | 4 | 4361.3 | 2.4 | 13.0 | 0.9 | 1.6 | 0.0 | 0.0 | 0.8 |
| | 6 | 8 | 3066.0 | 0.9 | 11.2 | 0.1 | 1.5 | 0.2 | 0.0 | 0.5 |
| | 7 | 10 | 1736.2 | 0.9 | 8.2 | 0.1 | 1.3 | 0.0 | 0.0 | 0.6 |
| | 10 | 8 | 2604.4 | 1.2 | 2.6 | 0.0 | 1.0 | 0.0 | 0.0 | 0.6 |
| | 12 | 10 | 773.5 | 2.4 | 1.5 | 0.0 | 1.2 | 0.0 | 0.0 | 0.5 |
| | 15 | 10 | 155.5 | 0.8 | 1.0 | 0.0 | 1.2 | 0.0 | 0.0 | 0.4 |
| Total | | 50 | 2116.2 | 1.4 | 6.3 | 0.2 | 1.3 | 0.0 | 0.0 | 0.6 |

Table 2: Aggregated average results on the instances in [16] with $m_i = 3$.

enumeration and verification of all the Eulerian circuits (see Section 5.3). These results support the practical importance of the method described in Appendix A. Although the non-preemptive problems are slightly different, our branch-and-cut algorithm outperforms the one in [7].

Among the 1800 instances taken from this article for the comparison, only four instances remain unsolved. All the others were solved quite efficiently, even when the initial load is bounded, the vehicle capacity is relatively small respect to the customer demands, and the preemption characteristic is forbidden.

## 9. Conclusions

This paper describes a new branch-and-cut approach to solve a pickup-and-delivery problem of one commodity where customers accept to be served through several visits. The algorithm can be adapted to force or relax assumptions on the vehicle load when starting the route, and also on the possibility of using customers to temporality store part of the vehicle load along the route. The branch-and-cut approach is based on solving an aggregated master model with integer variables counting the number of times each arc is traversed. Finding the vehicle load on each arc to certify the feasibility of a solution is a difficult problem addressed already in the literature. We introduce new alternatives for checking feasibility, one of them being the faster in our computation experiments. We also introduce a stronger valid inequality to eliminate infeasible solutions. With these elements, among other im-

| | | [7] | | | directed | | | undirected | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $z_i^*$ | | | check | | | | | |
| $n$ | $Q$ | Sol. | Gap | Time | Sol. | Gap | Time | $= 2$ | $= 3$ | $\geq 4$ | Fea. | Inf. | Time | Sol. | Gap | Time |
| 20 | 10 | 10 | | 0.3 | 10 | | 0.3 | 1.4 | 0.1 | 0.0 | 1.2 | 0.0 | 0.0 | 10 | | 0.2 |
| | 15 | 10 | | 0.3 | 10 | | 0.3 | 0.4 | 0.0 | 0.0 | 1.8 | 0.0 | 0.0 | 10 | | 0.2 |
| | 20 | 10 | | 0.1 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 25 | 10 | | 0.2 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 30 | 10 | | 2.0 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 35 | 10 | | 1.1 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 40 | 10 | | 1.2 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 45 | 10 | | 1.1 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| | 1000 | 10 | | 0.8 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| 30 | 10 | 10 | | 6.2 | 10 | | 2.7 | 2.5 | 0.0 | 0.0 | 3.2 | 0.0 | 0.0 | 10 | | 0.9 |
| | 15 | 10 | | 3.9 | 10 | | 1.4 | 1.0 | 0.0 | 0.0 | 2.4 | 0.0 | 0.0 | 10 | | 0.5 |
| | 20 | 10 | | 163.6 | 10 | | 0.6 | 0.6 | 0.0 | 0.0 | 1.5 | 0.0 | 0.0 | 10 | | 0.3 |
| | 25 | 10 | | 5.6 | 10 | | 0.4 | 0.1 | 0.0 | 0.0 | 1.1 | 0.0 | 0.0 | 10 | | 0.2 |
| | 30 | 10 | | 82.2 | 10 | | 0.5 | 0.1 | 0.0 | 0.0 | 1.6 | 0.2 | 0.0 | 10 | | 0.2 |
| | 35 | 10 | | 293.3 | 10 | | 0.4 | 0.2 | 0.0 | 0.0 | 1.8 | 0.4 | 0.0 | 10 | | 0.2 |
| | 40 | 10 | | 584.6 | 10 | | 0.6 | 0.3 | 0.0 | 0.0 | 2.5 | 13.9 | 0.0 | 10 | | 0.4 |
| | 45 | 10 | | 221.7 | 10 | | 0.6 | 0.2 | 0.0 | 0.0 | 1.6 | 12.6 | 0.0 | 10 | | 0.3 |
| | 1000 | 10 | | 190.2 | 10 | | 1.0 | 0.4 | 0.0 | 0.0 | 1.6 | 16.1 | 0.0 | 10 | | 0.3 |
| 40 | 10 | 10 | | 124.8 | 10 | | 21.4 | 2.8 | 0.0 | 0.0 | 4.6 | 0.0 | 0.0 | 10 | | 2.1 |
| | 15 | 10 | | 25.6 | 10 | | 4.1 | 0.3 | 0.0 | 0.0 | 3.1 | 0.0 | 0.0 | 10 | | 1.2 |
| | 20 | 10 | | 14.7 | 10 | | 1.6 | 0.3 | 0.0 | 0.0 | 2.2 | 0.0 | 0.0 | 10 | | 0.6 |
| | 25 | 9 | 0.03 | 723.9 | 10 | | 0.9 | 0.1 | 0.0 | 0.0 | 1.6 | 0.0 | 0.0 | 10 | | 0.5 |
| | 30 | 10 | | 36.6 | 10 | | 0.7 | 0.2 | 0.0 | 0.0 | 1.5 | 0.0 | 0.0 | 10 | | 0.3 |
| | 35 | 10 | | 38.7 | 10 | | 0.7 | 0.1 | 0.0 | 0.0 | 1.6 | 0.0 | 0.0 | 10 | | 0.3 |
| | 40 | 10 | | 70.7 | 10 | | 0.6 | 0.1 | 0.0 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.3 |
| | 45 | 10 | | 74.3 | 10 | | 0.6 | 0.1 | 0.0 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.3 |
| | 1000 | 10 | | 70.2 | 10 | | 0.6 | 0.1 | 0.0 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.3 |
| 50 | 10 | 9 | 0.79 | 1198.5 | 9 | 0.21 | 894.6 | 3.8 | 0.0 | 0.0 | 4.9 | 0.0 | 0.0 | 10 | | 60.1 |
| | 15 | 8 | 0.43 | 1970.1 | 9 | 0.29 | 1177.4 | 1.1 | 0.0 | 0.0 | 5.8 | 0.0 | 0.0 | 10 | | 18.1 |
| | 20 | 10 | | 295.5 | 10 | | 46.6 | 0.4 | 0.0 | 0.0 | 4.9 | 1.3 | 0.0 | 10 | | 3.7 |
| | 25 | 10 | | 272.8 | 10 | | 33.1 | 0.1 | 0.0 | 0.0 | 3.5 | 1.1 | 0.0 | 10 | | 1.7 |
| | 30 | 10 | | 177.4 | 10 | | 4.2 | 0.0 | 0.0 | 0.0 | 3.1 | 0.0 | 0.0 | 10 | | 1.0 |
| | 35 | 8 | 0.24 | 1461.1 | 10 | | 3.0 | 0.3 | 0.0 | 0.0 | 2.2 | 3.6 | 0.0 | 10 | | 1.0 |
| | 40 | 9 | 0.01 | 1408.9 | 10 | | 3.9 | 0.2 | 0.0 | 0.0 | 2.0 | 0.3 | 0.0 | 10 | | 0.9 |
| | 45 | 10 | | 1221.3 | 10 | | 3.0 | 0.2 | 0.0 | 0.0 | 2.7 | 0.0 | 0.0 | 10 | | 1.0 |
| | 1000 | 9 | 0.11 | 1909.8 | 10 | | 2.7 | 0.3 | 0.0 | 0.0 | 3.3 | 0.0 | 0.0 | 10 | | 0.9 |
| 60 | 10 | 6 | 1.24 | 3924.6 | 7 | 0.40 | 2887.4 | 5.4 | 0.0 | 0.0 | 10.8 | 0.0 | 0.0 | 10 | | 99.4 |
| | 15 | 8 | 0.51 | 1957.5 | 10 | | 874.4 | 1.2 | 0.0 | 0.0 | 5.2 | 0.0 | 0.0 | 10 | | 10.9 |
| | 20 | 10 | | 1285.0 | 10 | | 162.1 | 0.3 | 0.0 | 0.0 | 5.3 | 0.0 | 0.0 | 10 | | 6.4 |
| | 25 | 9 | 0.07 | 943.4 | 10 | | 50.3 | 0.1 | 0.0 | 0.0 | 4.3 | 2.0 | 0.0 | 10 | | 3.5 |
| | 30 | 9 | 0.13 | 1252.6 | 10 | | 61.6 | 0.2 | 0.0 | 0.0 | 3.6 | 0.4 | 0.0 | 10 | | 3.8 |
| | 35 | 9 | 0.13 | 1097.0 | 10 | | 18.1 | 0.2 | 0.0 | 0.0 | 3.9 | 0.0 | 0.0 | 10 | | 2.4 |
| | 40 | 7 | 0.22 | 2607.9 | 10 | | 11.5 | 0.2 | 0.0 | 0.0 | 3.5 | 8.9 | 0.0 | 10 | | 3.1 |
| | 45 | 7 | 0.15 | 2795.2 | 10 | | 12.1 | 0.5 | 0.0 | 0.0 | 3.5 | 4.0 | 0.0 | 10 | | 2.5 |
| | 1000 | 7 | 0.19 | 2816.4 | 10 | | 10.9 | 0.3 | 0.0 | 0.0 | 2.9 | 4.8 | 0.0 | 10 | | 2.1 |
| Total | | 424 | 0.09 | 696.3 | 445 | 0.02 | 140.0 | 0.6 | 0.0 | 0.0 | 2.7 | 1.5 | 0.0 | 450 | | 5.2 |

Table 3: Aggregated average results on the instances in [7] with preemption and $\alpha = 1$.

| | | [7] | | | directed | | | undirected | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $z_i^*$ | | | check | | | | | |
| $n$ | $Q$ | Sol. | Gap | Time | Sol. | Gap | Time | 2 | 3 | $\geq 4$ | Fea. | Inf. | Time | Sol. | Gap | Time |
| 20 | 10 | 10 | | 0.5 | 10 | | 0.6 | 5.7 | 7.4 | 0.2 | 1.6 | 0.0 | 0.0 | 10 | | 0.5 |
| | 15 | 10 | | 0.3 | 10 | | 0.5 | 8.8 | 0.9 | 0.0 | 1.7 | 0.0 | 0.0 | 10 | | 0.4 |
| | 20 | 10 | | 0.4 | 10 | | 0.4 | 8.2 | 0.1 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.3 |
| | 25 | 10 | | 0.4 | 10 | | 0.4 | 3.7 | 0.0 | 0.0 | 1.6 | 0.0 | 0.0 | 10 | | 0.3 |
| | 30 | 10 | | 0.4 | 10 | | 0.3 | 1.4 | 0.1 | 0.0 | 1.2 | 0.0 | 0.0 | 10 | | 0.2 |
| | 35 | 10 | | 0.3 | 10 | | 0.4 | 1.3 | 0.0 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.2 |
| | 40 | 10 | | 0.3 | 10 | | 0.3 | 0.8 | 0.0 | 0.0 | 1.2 | 0.1 | 0.0 | 10 | | 0.2 |
| | 45 | 10 | | 0.3 | 10 | | 0.3 | 0.4 | 0.0 | 0.0 | 1.8 | 0.0 | 0.0 | 10 | | 0.2 |
| | 1000 | 10 | | 0.9 | 10 | | 0.2 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.1 |
| 30 | 10 | 10 | | 153.8 | 10 | | 90.1 | 8.8 | 10.2 | 0.4 | 3.0 | 0.0 | 0.0 | 10 | | 1.8 |
| | 15 | 10 | | 65.4 | 10 | | 5.2 | 13.6 | 0.8 | 0.0 | 3.2 | 0.0 | 0.0 | 10 | | 1.1 |
| | 20 | 10 | | 8.2 | 10 | | 1.6 | 11.8 | 0.0 | 0.0 | 2.8 | 0.0 | 0.0 | 10 | | 0.8 |
| | 25 | 10 | | 10.3 | 10 | | 2.3 | 6.4 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 10 | | 0.9 |
| | 30 | 10 | | 9.9 | 10 | | 2.8 | 2.5 | 0.0 | 0.0 | 3.2 | 0.0 | 0.0 | 10 | | 0.9 |
| | 35 | 10 | | 9.4 | 10 | | 4.2 | 2.6 | 0.0 | 0.0 | 3.7 | 0.0 | 0.0 | 10 | | 0.9 |
| | 40 | 10 | | 5.9 | 10 | | 1.4 | 1.2 | 0.0 | 0.0 | 2.2 | 0.0 | 0.0 | 10 | | 0.6 |
| | 45 | 10 | | 3.0 | 10 | | 1.4 | 1.0 | 0.0 | 0.0 | 2.4 | 0.0 | 0.0 | 10 | | 0.5 |
| | 1000 | 10 | | 221.3 | 10 | | 1.0 | 0.4 | 0.0 | 0.0 | 1.6 | 16.1 | 0.0 | 10 | | 0.3 |
| 40 | 10 | 10 | | 235.4 | 10 | | 12.5 | 12.8 | 11.3 | 0.5 | 4.7 | 0.1 | 0.0 | 10 | | 3.5 |
| | 15 | 10 | | 28.8 | 10 | | 5.0 | 15.9 | 0.5 | 0.0 | 2.8 | 0.0 | 0.0 | 10 | | 1.6 |
| | 20 | 10 | | 62.2 | 10 | | 7.7 | 12.5 | 0.1 | 0.0 | 3.7 | 0.0 | 0.0 | 10 | | 2.1 |
| | 25 | 10 | | 177.4 | 10 | | 29.4 | 6.4 | 0.1 | 0.0 | 3.7 | 0.0 | 0.0 | 10 | | 3.3 |
| | 30 | 10 | | 108.3 | 10 | | 22.8 | 2.8 | 0.0 | 0.0 | 4.6 | 0.0 | 0.0 | 10 | | 2.3 |
| | 35 | 10 | | 304.7 | 10 | | 36.8 | 1.6 | 0.1 | 0.0 | 4.5 | 0.0 | 0.0 | 10 | | 5.1 |
| | 40 | 10 | | 21.7 | 10 | | 4.3 | 1.2 | 0.0 | 0.0 | 2.7 | 0.0 | 0.0 | 10 | | 1.2 |
| | 45 | 10 | | 25.7 | 10 | | 4.4 | 0.3 | 0.0 | 0.0 | 3.1 | 0.0 | 0.0 | 10 | | 1.3 |
| | 1000 | 10 | | 80.9 | 10 | | 0.6 | 0.1 | 0.0 | 0.0 | 1.4 | 0.0 | 0.0 | 10 | | 0.3 |
| 50 | 10 | 7 | 0.99 | 2693.4 | 9 | 0.16 | 1023.8 | 17.0 | 15.1 | 0.9 | 8.5 | 0.7 | 3.7 | 10 | | 53.6 |
| | 15 | 10 | | 1702.2 | 10 | | 469.1 | 21.8 | 0.6 | 0.2 | 5.0 | 0.0 | 0.0 | 10 | | 20.4 |
| | 20 | 7 | 0.90 | 3085.4 | 7 | 0.23 | 2270.0 | 17.5 | 0.1 | 0.0 | 10.3 | 0.0 | 0.1 | 10 | | 82.1 |
| | 25 | 8 | 0.59 | 2024.5 | 10 | | 325.7 | 7.9 | 0.2 | 0.0 | 7.3 | 0.0 | 0.0 | 10 | | 32.9 |
| | 30 | 9 | 0.46 | 1345.2 | 9 | 0.21 | 893.3 | 3.8 | 0.0 | 0.0 | 4.9 | 0.0 | 0.0 | 10 | | 60.1 |
| | 35 | 5 | 1.27 | 4212.4 | 9 | 0.01 | 2114.2 | 3.6 | 0.1 | 0.0 | 9.0 | 0.0 | 0.1 | 10 | | 49.4 |
| | 40 | 6 | 0.45 | 3545.4 | 10 | | 1821.3 | 2.3 | 0.0 | 0.0 | 6.5 | 0.4 | 0.0 | 10 | | 24.9 |
| | 45 | 8 | 0.23 | 2057.8 | 9 | 0.30 | 1176.4 | 1.1 | 0.0 | 0.0 | 5.8 | 0.0 | 0.0 | 10 | | 18.3 |
| | 1000 | 9 | 0.10 | 1939.0 | 10 | | 2.7 | 0.3 | 0.0 | 0.0 | 3.3 | 0.0 | 0.0 | 10 | | 0.9 |
| 60 | 10 | 6 | 2.83 | 3718.0 | 8 | 0.15 | 1812.5 | 18.5 | 19.5 | 1.1 | 8.2 | 0.1 | 0.5 | 10 | | 118.3 |
| | 15 | 7 | 2.73 | 2932.6 | 7 | 0.25 | 2454.6 | 27.8 | 1.0 | 0.0 | 9.5 | 0.0 | 0.0 | 10 | | 568.6 |
| | 20 | 7 | 0.39 | 3772.6 | 8 | 0.08 | 1978.3 | 22.8 | 0.2 | 0.0 | 6.5 | 0.0 | 0.1 | 10 | | 47.8 |
| | 25 | 6 | 2.63 | 3636.1 | 8 | 3.84 | 2403.7 | 11.7 | 0.0 | 0.0 | 7.8 | 0.0 | 0.0 | 10 | | 89.2 |
| | 30 | 6 | 2.88 | 4702.4 | 7 | 0.40 | 2890.2 | 5.4 | 0.0 | 0.0 | 10.8 | 0.0 | 0.0 | 10 | | 99.7 |
| | 35 | 6 | 2.42 | 4795.7 | 8 | 0.42 | 2249.9 | 4.2 | 0.0 | 0.0 | 9.2 | 0.0 | 0.0 | 10 | | 81.4 |
| | 40 | 7 | 0.46 | 3830.0 | 10 | | 955.4 | 1.6 | 0.0 | 0.0 | 8.0 | 0.2 | 0.0 | 10 | | 29.8 |
| | 45 | 8 | 0.26 | 2223.5 | 10 | | 899.7 | 1.2 | 0.0 | 0.0 | 5.2 | 0.0 | 0.0 | 10 | | 11.1 |
| | 1000 | 7 | 0.18 | 2940.4 | 10 | | 11.9 | 0.3 | 0.0 | 0.0 | 2.9 | 4.8 | 0.0 | 10 | | 2.3 |
| Total | | 399 | 0.44 | 1259.8 | 429 | 0.13 | 577.6 | 6.7 | 1.5 | 0.1 | 4.4 | 0.5 | 0.1 | 450 | | 31.6 |

Table 4: Aggregated average results on the instances in [7] with preemption and $\alpha = 3$.

| | | [7] | | | | directed | | | undirected | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | $z_i^*$ | | | check | | | | | | |
| $n$ | $Q$ | Sol. | Gap | Gap' | Time | Sol. | Gap | Time | $=2$ | $=3$ | $\geq 4$ | Fea. | Inf. | Time | Sol. | Gap | Gap' | Time |
| 20 | 10 | 10 | | 0.29 | 3.6 | 10 | | 0.4 | 1.7 | 0.0 | 0.0 | 1.2 | 2.2 | 0.0 | 10 | | 0.60 | 0.2 |
| | 15 | 10 | | 1.05 | 9.5 | 10 | | 0.3 | 0.5 | 0.0 | 0.0 | 2.8 | 0.3 | 0.0 | 10 | | 1.56 | 0.2 |
| | 20 | 10 | | 0.34 | 0.4 | 10 | | 0.2 | 0.2 | 0.0 | 0.0 | 1.2 | 0.4 | 0.0 | 10 | | 1.31 | 0.1 |
| | 25 | 10 | | 0.70 | 0.6 | 10 | | 0.2 | 0.1 | 0.0 | 0.0 | 1.2 | 0.5 | 0.0 | 10 | | 1.26 | 0.1 |
| | 30 | 10 | | 0.44 | 2.2 | 10 | | 0.2 | 0.1 | 0.0 | 0.0 | 1.2 | 0.1 | 0.0 | 10 | | 1.00 | 0.1 |
| | 35 | 10 | | 1.14 | 89.6 | 10 | | 1.3 | 0.1 | 0.0 | 0.0 | 1.4 | 51.6 | 0.0 | 10 | | 1.70 | 0.6 |
| | 40 | 10 | | 1.14 | 102.1 | 10 | | 1.2 | 0.2 | 0.0 | 0.0 | 1.4 | 46.8 | 0.0 | 10 | | 1.70 | 0.5 |
| | 45 | 10 | | 1.14 | 94.6 | 10 | | 1.1 | 0.0 | 0.2 | 0.0 | 1.6 | 49.6 | 0.0 | 10 | | 1.70 | 0.5 |
| | 1000 | 10 | | 1.14 | 93.0 | 10 | | 0.9 | 0.0 | 0.1 | 0.1 | 1.6 | 56.2 | 0.0 | 10 | | 1.70 | 0.6 |
| 30 | 10 | 10 | | 0.11 | 44.6 | 10 | | 1.9 | 3.2 | 0.0 | 0.0 | 2.8 | 2.6 | 0.0 | 10 | | 0.14 | 0.7 |
| | 15 | 10 | | 0.04 | 4.5 | 10 | | 1.0 | 1.2 | 0.0 | 0.0 | 1.3 | 0.0 | 0.0 | 10 | | 0.08 | 0.4 |
| | 20 | 10 | | 0.02 | 139.0 | 10 | | 0.5 | 0.7 | 0.0 | 0.0 | 1.5 | 0.2 | 0.0 | 10 | | 0.52 | 0.2 |
| | 25 | 10 | | 0.05 | 4.0 | 10 | | 0.3 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 10 | | 0.65 | 0.2 |
| | 30 | 10 | | 0.33 | 64.2 | 10 | | 1.2 | 0.5 | 0.0 | 0.0 | 1.8 | 8.9 | 0.0 | 10 | | 1.41 | 0.3 |
| | 35 | 10 | | 0.46 | 640.7 | 10 | | 1.0 | 0.2 | 0.0 | 0.0 | 1.8 | 5.4 | 0.0 | 10 | | 1.36 | 0.2 |
| | 40 | 10 | | 0.33 | 182.7 | 10 | | 1.0 | 0.3 | 0.0 | 0.0 | 1.8 | 4.5 | 0.0 | 10 | | 1.46 | 0.2 |
| | 45 | 10 | | 0.33 | 145.7 | 10 | | 4.6 | 0.5 | 0.0 | 0.0 | 1.2 | 4.3 | 0.0 | 10 | | 1.46 | 0.2 |
| | 1000 | 10 | | 0.33 | 125.4 | 10 | | 4.7 | 0.5 | 0.0 | 0.0 | 1.2 | 4.3 | 0.0 | 10 | | 1.46 | 0.2 |
| 40 | 10 | 8 | 0.39 | 0.55 | 1538.5 | 10 | | 17.0 | 2.8 | 0.0 | 0.0 | 4.4 | 5.4 | 0.0 | 10 | | 0.86 | 2.3 |
| | 15 | 10 | | 0.30 | 444.6 | 10 | | 2.8 | 0.9 | 0.0 | 0.0 | 2.1 | 0.5 | 0.0 | 10 | | 0.73 | 0.9 |
| | 20 | 9 | 0.14 | 0.61 | 1241.5 | 10 | | 4.3 | 0.9 | 0.0 | 0.0 | 3.4 | 20.3 | 0.0 | 10 | | 1.75 | 1.0 |
| | 25 | 9 | 0.15 | 0.86 | 1132.4 | 10 | | 7.1 | 0.7 | 0.0 | 0.0 | 4.3 | 52.1 | 0.0 | 10 | | 2.08 | 1.9 |
| | 30 | 9 | 0.15 | 0.85 | 831.2 | 10 | | 6.0 | 0.8 | 0.1 | 0.0 | 3.0 | 31.2 | 0.0 | 10 | | 2.10 | 1.1 |
| | 35 | 9 | 0.15 | 0.77 | 798.1 | 10 | | 2.3 | 0.8 | 0.0 | 0.0 | 2.7 | 50.5 | 0.0 | 10 | | 2.23 | 1.5 |
| | 40 | 9 | 0.16 | 0.86 | 855.0 | 10 | | 2.4 | 0.9 | 0.0 | 0.0 | 4.2 | 74.9 | 0.0 | 10 | | 2.39 | 1.9 |
| | 45 | 9 | 0.16 | 0.86 | 859.5 | 10 | | 2.9 | 1.2 | 0.0 | 0.0 | 3.1 | 54.6 | 0.0 | 10 | | 2.38 | 1.4 |
| | 1000 | 9 | 0.16 | 0.86 | 892.5 | 10 | | 2.9 | 1.1 | 0.0 | 0.0 | 3.0 | 53.0 | 0.0 | 10 | | 2.38 | 1.3 |
| 50 | 10 | 8 | 0.12 | N/A | 2211.4 | 9 | 0.12 | 838.5 | 3.5 | 0.1 | 0.0 | 6.4 | 15.1 | 0.0 | 10 | | 0.22 | 37.0 |
| | 15 | 6 | 5.26 | N/A | 3693.6 | 10 | | 918.9 | 2.0 | 0.0 | 0.0 | 5.9 | 135.4 | 0.0 | 10 | | 0.49 | 30.7 |
| | 20 | 9 | 0.13 | N/A | 1288.8 | 10 | | 41.8 | 1.0 | 0.0 | 0.0 | 5.0 | 34.4 | 0.0 | 10 | | 0.72 | 4.7 |
| | 25 | 9 | 0.21 | N/A | 1326.7 | 10 | | 12.8 | 0.6 | 0.0 | 0.0 | 4.8 | 18.1 | 0.0 | 10 | | 0.51 | 2.6 |
| | 30 | 8 | 0.37 | N/A | 2719.3 | 10 | | 8.7 | 0.3 | 0.0 | 0.0 | 3.6 | 18.9 | 0.0 | 10 | | 0.99 | 2.5 |
| | 35 | 7 | 0.60 | N/A | 3265.8 | 9 | 0.04 | 723.4 | 0.6 | 0.0 | 0.0 | 4.2 | 498.8 | 0.0 | 10 | | 0.88 | 294.8 |
| | 40 | 8 | 0.32 | N/A | 2115.6 | 10 | | 4.0 | 0.1 | 0.0 | 0.0 | 2.5 | 15.1 | 4.8 | 10 | | 0.76 | 6.3 |
| | 45 | 8 | 0.35 | N/A | 2399.3 | 10 | | 7.1 | 0.4 | 0.0 | 0.0 | 2.6 | 27.9 | 0.0 | 10 | | 0.76 | 1.6 |
| | 1000 | 6 | 0.84 | N/A | 4236.3 | 9 | 0.17 | 729.0 | 0.3 | 0.0 | 0.0 | 3.4 | 804.1 | 0.0 | 10 | | 1.31 | 617.1 |
| 60 | 10 | 5 | 8.45 | N/A | 4272.5 | 6 | 0.67 | 3240.3 | 6.2 | 0.1 | 0.0 | 7.9 | 208.7 | 2.0 | 10 | | 0.33 | 426.3 |
| | 15 | 6 | 7.53 | N/A | 3725.4 | 10 | | 934.7 | 1.7 | 0.0 | 0.0 | 5.7 | 36.7 | 0.0 | 10 | | 0.44 | 15.1 |
| | 20 | 8 | 0.17 | N/A | 2577.4 | 10 | | 232.4 | 0.3 | 0.0 | 0.0 | 3.7 | 10.5 | 0.0 | 10 | | 0.62 | 4.1 |
| | 25 | 7 | 0.19 | N/A | 2307.8 | 10 | | 82.8 | 0.7 | 0.0 | 0.0 | 4.6 | 15.3 | 0.0 | 10 | | 0.84 | 3.1 |
| | 30 | 6 | 0.75 | N/A | 3196.2 | 10 | | 79.6 | 0.5 | 0.0 | 0.0 | 4.2 | 12.3 | 0.0 | 10 | | 0.93 | 6.3 |
| | 35 | 5 | 1.05 | N/A | 4126.4 | 10 | | 185.9 | 0.3 | 0.0 | 0.0 | 4.5 | 24.1 | 0.0 | 10 | | 1.05 | 7.3 |
| | 40 | 4 | 1.13 | N/A | 4861.3 | 10 | | 56.4 | 1.1 | 0.0 | 0.0 | 4.5 | 124.0 | 0.0 | 10 | | 1.13 | 20.5 |
| | 45 | 3 | 1.51 | N/A | 5127.3 | 8 | 0.33 | 1453.8 | 1.2 | 0.0 | 0.0 | 4.7 | 1293.1 | 0.1 | 8 | 0.21 | 1.44 | 1444.2 |
| | 1000 | 2 | 1.21 | N/A | 5757.9 | 10 | | 90.5 | 0.5 | 0.0 | 0.0 | 4.6 | 130.6 | 0.0 | 10 | | 1.18 | 19.3 |
| Total | | 376 | 0.70 | | 1545.5 | 441 | 0.03 | 215.8 | 0.9 | 0.0 | 0.0 | 3.1 | 89.0 | 0.2 | 448 | 0.00 | 1.17 | 65.8 |

Table 5: Aggregated average results on the instances in [7] without preemption and $\alpha = 1$.

| | | [7] | | | | directed | | | $z_i^*$ | | | undirected check | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $Q$ | Sol. | Gap | Gap' | Time | Sol. | Gap | Time | $=2$ | $=3$ | $\geq 4$ | Fea. | Inf. | Time | Sol. | Gap | Gap' | Time |
| 20 | 10 | 10 | | | 0.6 | 10 | | 0.6 | 5.8 | 7.4 | 0.1 | 1.8 | 0.1 | 0.0 | 10 | | 0.01 | 0.5 |
| | 15 | 10 | | 0.06 | 0.7 | 10 | | 0.4 | 8.8 | 1.0 | 0.0 | 1.7 | 0.0 | 0.0 | 10 | | 0.06 | 0.3 |
| | 20 | 10 | | 0.08 | 0.7 | 10 | | 0.4 | 8.1 | 0.1 | 0.0 | 1.2 | 0.0 | 0.0 | 10 | | 0.21 | 0.3 |
| | 25 | 10 | | | 0.3 | 10 | | 0.4 | 3.9 | 0.0 | 0.0 | 1.1 | 0.1 | 0.0 | 10 | | 0.22 | 0.3 |
| | 30 | 10 | | 0.29 | 3.3 | 10 | | 0.4 | 1.7 | 0.0 | 0.0 | 1.2 | 2.2 | 0.0 | 10 | | 0.60 | 0.2 |
| | 35 | 10 | | 0.42 | 10.2 | 10 | | 0.4 | 1.3 | 0.0 | 0.0 | 1.6 | 0.5 | 0.0 | 10 | | 0.74 | 0.2 |
| | 40 | 10 | | 0.52 | 8.3 | 10 | | 0.3 | 0.9 | 0.0 | 0.0 | 1.8 | 3.6 | 0.0 | 10 | | 0.91 | 0.2 |
| | 45 | 10 | | 1.05 | 8.7 | 10 | | 0.3 | 0.5 | 0.0 | 0.0 | 2.8 | 0.3 | 0.0 | 10 | | 1.56 | 0.2 |
| | 1000 | 10 | | 1.14 | 91.9 | 10 | | 0.9 | 0.0 | 0.1 | 0.1 | 1.6 | 56.2 | 0.0 | 10 | | 1.70 | 0.6 |
| 30 | 10 | 10 | | 0.01 | 185.9 | 10 | | 24.1 | 8.6 | 10.3 | 0.3 | 4.0 | 0.9 | 5.7 | 10 | | 0.01 | 7.3 |
| | 15 | 10 | | | 60.1 | 9 | 0.01 | 723.1 | 13.4 | 0.9 | 0.0 | 3.6 | 403.0 | 13.1 | 10 | | 0.01 | 45.7 |
| | 20 | 10 | | | 4.9 | 10 | | 1.6 | 11.7 | 0.0 | 0.0 | 3.0 | 0.7 | 0.0 | 10 | | | 0.8 |
| | 25 | 10 | | 0.06 | 67.8 | 10 | | 140.2 | 7.1 | 0.2 | 0.0 | 3.2 | 24.1 | 0.0 | 10 | | 0.27 | 1.1 |
| | 30 | 10 | | 0.11 | 44.6 | 10 | | 2.0 | 3.2 | 0.0 | 0.0 | 2.8 | 2.6 | 0.0 | 10 | | 0.14 | 0.8 |
| | 35 | 10 | | | 8.7 | 10 | | 2.0 | 2.7 | 0.0 | 0.0 | 2.2 | 0.7 | 0.0 | 10 | | 0.07 | 0.6 |
| | 40 | 9 | 0.06 | 0.28 | 723.7 | 10 | | 2.4 | 1.6 | 0.0 | 0.0 | 2.5 | 5.1 | 0.0 | 10 | | 0.44 | 0.6 |
| | 45 | 10 | | 0.04 | 4.5 | 10 | | 1.1 | 1.2 | 0.0 | 0.0 | 1.3 | 0.0 | 0.0 | 10 | | 0.08 | 0.5 |
| | 1000 | 10 | | 0.33 | 118.6 | 10 | | 4.6 | 0.5 | 0.0 | 0.0 | 1.2 | 4.3 | 0.0 | 10 | | 1.46 | 0.2 |
| 40 | 10 | 9 | | 0.01 | 1007.1 | 10 | | 28.6 | 12.8 | 10.9 | 0.7 | 5.1 | 3.6 | 21.4 | 10 | | 0.01 | 24.7 |
| | 15 | 10 | | | 97.7 | 10 | | 4.0 | 15.6 | 0.6 | 0.0 | 3.2 | 0.8 | 0.0 | 10 | | 0.00 | 1.4 |
| | 20 | 7 | 0.11 | 0.20 | 2186.8 | 10 | | 4.8 | 13.3 | 0.0 | 0.0 | 4.0 | 0.5 | 0.0 | 10 | | 0.18 | 1.8 |
| | 25 | 8 | 0.02 | 0.08 | 1618.7 | 10 | | 171.0 | 7.5 | 0.1 | 0.0 | 4.2 | 31.1 | 0.0 | 10 | | 0.43 | 2.7 |
| | 30 | 8 | 0.39 | 0.55 | 1538.2 | 10 | | 16.9 | 2.8 | 0.0 | 0.0 | 4.4 | 5.4 | 0.0 | 10 | | 0.86 | 2.3 |
| | 35 | 7 | 0.42 | 0.63 | 2333.3 | 9 | 0.02 | 800.2 | 2.1 | 0.0 | 0.0 | 6.0 | 182.4 | 0.1 | 10 | | 0.71 | 21.1 |
| | 40 | 9 | 0.22 | 0.55 | 1179.0 | 10 | | 4.9 | 1.5 | 0.0 | 0.0 | 2.7 | 18.5 | 0.0 | 10 | | 0.72 | 1.6 |
| | 45 | 10 | | 0.30 | 450.0 | 10 | | 3.0 | 0.9 | 0.0 | 0.0 | 2.1 | 0.5 | 0.0 | 10 | | 0.73 | 1.0 |
| | 1000 | 9 | 0.16 | 0.86 | 893.6 | 10 | | 2.9 | 1.1 | 0.0 | 0.0 | 3.0 | 53.0 | 0.0 | 10 | | 2.38 | 1.4 |
| 50 | 10 | 6 | 0.08 | N/A | 3461.0 | 9 | 0.10 | 975.4 | 17.0 | 14.9 | 0.8 | 8.2 | 17.8 | 108.7 | 10 | | 0.01 | 160.1 |
| | 15 | 7 | | N/A | 2702.2 | 9 | 0.01 | 788.6 | 22.1 | 0.6 | 0.1 | 6.3 | 348.4 | 618.3 | 9 | 0.01 | 0.02 | 727.4 |
| | 20 | 6 | 2.51 | N/A | 3151.7 | 9 | 0.23 | 1681.1 | 17.2 | 0.2 | 0.0 | 8.4 | 5.8 | 0.8 | 10 | | 0.03 | 34.9 |
| | 25 | 8 | 1.77 | N/A | 2329.1 | 9 | 0.22 | 983.3 | 9.0 | 0.2 | 0.0 | 6.0 | 10.7 | 22.0 | 10 | | 0.04 | 44.4 |
| | 30 | 8 | 4.68 | N/A | 2300.2 | 9 | 0.12 | 839.4 | 3.5 | 0.1 | 0.0 | 6.4 | 15.1 | 0.0 | 10 | | 0.22 | 37.0 |
| | 35 | 5 | 1.55 | N/A | 4067.3 | 9 | 0.16 | 1302.4 | 3.3 | 0.1 | 0.0 | 6.2 | 884.6 | 1.2 | 9 | 0.04 | 0.29 | 742.6 |
| | 40 | 7 | 1.58 | N/A | 3529.6 | 10 | | 381.0 | 3.0 | 0.0 | 0.0 | 5.7 | 21.5 | 0.0 | 10 | | 0.23 | 19.3 |
| | 45 | 6 | 5.26 | N/A | 3769.5 | 10 | | 935.8 | 2.0 | 0.0 | 0.0 | 5.9 | 135.4 | 0.0 | 10 | | 0.49 | 31.0 |
| | 1000 | 6 | 0.85 | N/A | 4347.9 | 9 | 0.17 | 729.0 | 0.3 | 0.0 | 0.0 | 3.4 | 804.1 | 0.0 | 10 | | 1.31 | 611.1 |
| 60 | 10 | 6 | 2.74 | N/A | 3851.6 | 8 | 0.11 | 1888.8 | 18.4 | 19.4 | 1.2 | 11.3 | 26.0 | 186.5 | 10 | | 0.04 | 293.3 |
| | 15 | 6 | 0.48 | N/A | 3562.7 | 7 | 0.21 | 2500.7 | 27.9 | 0.8 | 0.0 | 9.2 | 3.6 | 11.1 | 10 | | 0.04 | 53.3 |
| | 20 | 5 | 2.99 | N/A | 4638.2 | 9 | 0.05 | 1379.3 | 22.9 | 0.4 | 0.0 | 8.7 | 20.5 | 64.4 | 10 | | 0.04 | 126.0 |
| | 25 | 5 | 13.54 | N/A | 3933.8 | 9 | 0.07 | 1284.7 | 12.6 | 0.0 | 0.0 | 8.0 | 6.2 | 20.2 | 10 | | 0.08 | 83.5 |
| | 30 | 5 | 8.50 | N/A | 4347.2 | 6 | 0.66 | 3236.4 | 6.2 | 0.1 | 0.0 | 7.9 | 208.7 | 1.9 | 10 | | 0.33 | 420.7 |
| | 35 | 6 | 4.63 | N/A | 4523.9 | 7 | 0.88 | 3311.4 | 5.2 | 0.0 | 0.0 | 10.4 | 469.0 | 0.1 | 10 | | 0.30 | 530.0 |
| | 40 | 6 | 1.97 | N/A | 3534.8 | 9 | 0.31 | 1094.6 | 2.0 | 0.0 | 0.0 | 7.3 | 7.9 | 0.0 | 10 | | 0.31 | 16.1 |
| | 45 | 6 | 0.39 | N/A | 3730.1 | 10 | | 934.6 | 1.7 | 0.0 | 0.0 | 5.7 | 36.7 | 0.0 | 10 | | 0.44 | 15.0 |
| | 1000 | 2 | 1.19 | N/A | 5759.2 | 10 | | 91.2 | 0.5 | 0.0 | 0.0 | 4.6 | 130.6 | 0.0 | 10 | | 1.18 | 19.5 |
| Total | | 362 | 1.25 | | 1782.0 | 426 | 0.07 | 584.0 | 7.0 | 1.5 | 0.1 | 4.5 | 87.8 | 23.9 | 448 | 0.00 | 0.44 | 90.8 |

Table 6: Aggregated average results on the instances in [7] without preemption and $\alpha = 3$.

provements, the branch-and-cut algorithm described in this paper is quite satisfactory to find optimal solutions of instances with up to 60 customers, no matter whether the initial vehicle load is fixed and whether some or all customers can be used as buffers to temporarily store commodity.

## References


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

[2] Archetti, C., Speranza, M. G., 2012. Vehicle routing problems with split deliveries. International Transactions in Operational Research 19 (1-2), 3–22.

[3] Chemla, D., Meunier, F., Wolfler-Calvo, R., 2013. Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization 10 (2), 120–146.

[4] Cruz, F., Bruck, B.P., Subramanian, A., Iori, M., 2017. A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. Computers and Operations Research 79, 19–33.

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

[6] Dell'Amico, M., Iori, M., Novellani, S., Stützle, T., 2016. A Destroy and Repair Algorithm for the Bike sharing Rebalancing Problem. Computers and Operations Research 71, 149–162.

[7] Erdogan, G., Battarra, M., Wolfler Calvo, R., 2015. An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. European Journal of Operational Research 245, 667–679.

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

[9] Hernández-Pérez, H., Salazar-González, J. J., 2004. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. Discrete Applied Mathematics 145, 126–139.

[10] Hernández-Pérez, H., Salazar-González, J. J., 2004. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. Transportation Science 38, 245–255.

[11] Hernández-Pérez, H., Salazar-González, J. J., 2018. An Exact Algorithm for the Split-Demand One-Commodity Pickup-and-delivery Travelling Salesman Problem. In: Lee, J., Rinaldi, G., Mahjoub, A. R. (Eds.), Lectures Notes in Computer Science 10856, 241–252.

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

[13] Quilliot, A., Sarbinowski, A., Toussaint, H., 2020. Vehicle driven approaches for non preemptive vehicle relocation with integrated quality criterion in a vehicle sharing system. Annals of Operations Research, in press.

[14] Rainer-Harbach, M., Papazek, P., Hu, B., Riadl, G. R., 2015. Balancing bicycle sharing systems: A variable neighbourhood approach. Journal of Global Optimization 63, 597–629.

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

[16] Salazar-González, J. J., Santos-Hernández, B., 2015. The split-demand one-commodity pickup-and-delivery travelling salesman problem. Transportation Research Part B: Methodological 75, 58–73.

[17] Zhao, F., Li, S., Sun, J., Mei, D., 2009. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. Computers and Industrial Engineering 56 (4), 1642–1648.

## Appendix A. A new algorithm to check the feasibility of a circuit

This section describes a new algorithm to check the SD1PDTSP feasibility of a circuit. A summary of the algorithm and new notation is presented in Appendix A.1. Subroutines are described from Appendix A.2 to Appendix A.5. The overall algorithm is detailed in Appendix A.6. Finally, Appendix A.7 describes the algorithm for the non-preemptive case.

### *Appendix A.1. Summary and notation*

The algorithm sets a possible demand interval for each visit and a possible load interval for the vehicle when it leaves each vertex. Iteratively, it sets a demand interval and a load interval for each visit. The setting of a certain interval (for the demand and/or the load) may readjust the intervals of previous positions in the circuit. If this readjustment determines that the lower bound is greater than the upper bound of an interval, the algorithm concludes that the circuit is infeasible. Otherwise, the algorithm applies two procedures to determine whether the solution is feasible or infeasible. The first procedure is a heuristic approach, and the second procedure is an exact approach.

The readjustment is a key phase in the algorithm. This readjustment can happen in different places of the circuit. When a new position is used to set the demand and load intervals, it can modify the load interval of a previous position. Even if a position corresponds to the second or subsequent visits to a vertex, the load interval of the vehicle at the previous visits can be modified. Moreover, a readjustment in an intermediate position of the circuit can produce changes in the demand and load intervals of its next positions. Note that readjustments always shrink the intervals.

A solution for the SD1PDTSP can be represented by a sequence of visits to locations, starting and ending at the depot. This sequence is a circuit $C = (v_0, \ldots, v_k)$ with $v_r \in I$ for $r = 0, \ldots, k$ and $v_0 = v_k = 1$ where $k$ is the total number of visits to all locations. We denote by $g_r$ the demand of the location in the visit $v_r$ for $r = 0, \ldots, k$. Positive or negative values of $g_r$ indicate the amount delivered or picked up, respectively. In addition, we denote by $f_r$ the load of the vehicle leaving $v_r$ for $r = 0, \ldots, k$. Given a circuit $C$, $z_i^*$ refers to the number of visits to location $i$ for $i \in I$. Expanding the notation of variables $g_r$, we denote

by $g_{i_j}$ the demand of location $i$ in its $j$-th visit, for $i \in I$ and $j = 1, \ldots, z_i^*$. Sometimes, we also write $v_r$ as $i_j$ if $v_r$ corresponds to the $j$-th visit to location $i$.

From equations (6)–(10), a circuit $C = (v_0, \ldots, v_k)$ is SD1PDTSP feasible if there exist values $g_r$ and $f_r$ satisfying:

$$0 \leq f_r \leq Q \qquad \text{for all } r = 0, \ldots, k-1 \tag{A.1}$$

$$f_r = f_{r-1} - g_r \qquad \text{for all } r = 1, \ldots, k-1 \tag{A.2}$$

$$\sum_{1 \leq j \leq z_i^*} g_{i_j} = d_i \qquad \text{for all } i \in I \tag{A.3}$$

$$0 \leq p_i + \sum_{1 \leq j' \leq j} g_{i_{j'}} \leq q_i \qquad \text{for all } i \in I \ , \ j = 1, \ldots, z_i^* - 1. \tag{A.4}$$

Inequalities (A.1) force that the load of the vehicle is always between 0 and $Q$. Equalities (A.2) enforce the flow conservation for the product in each visit. Equalities (A.3) impose that the total quantity required (or offered) by $i$ is $d_i$. Finally, inequalities (A.4) force that the cumulative amount of product at location $i$ is never neither negative nor greater than its maximum stock capacity $q_i$.

When preemption is not allowed, inequalities (A.4) must be replaced by inequalities:

$$g_{i_j} \geq 0 \qquad \text{for all } i \in I : d_i \geq 0 \ , \ \ j = 1, \ldots, z_i^* \tag{A.5}$$

$$g_{i_j} \leq 0 \qquad \text{for all } i \in I : d_i < 0 \ , \ \ j = 1, \ldots, z_i^*. \tag{A.6}$$

We denoted by $\underline{g}_r$ and $\overline{g}_r$ a lower bound and an upper bound of $g_r$ for $r = 0, \ldots, k$. Similarly, we denoted by $\underline{f}_r$ and $\overline{f}_r$ a lower bound and an upper bound of $f_r$ for $r = 0, \ldots, k$.

*Appendix A.2. Setting the demand and load intervals*

We address the first setting to the demand and load intervals. Given a circuit $C = (v_0, \ldots, v_k)$, we assume that we have the demand and load intervals for position from 0 to $r-1$ and $v_r$ corresponds with the $j$-th visit to location $i$ (i.e, $v_r = i_j$).

From Constraints (A.1)–(A.4) we obtain three lower bounds for $g_r$, and $\underline{g}_r$ can be set to the maximum of them:

$$\underline{g}_r := \max \{\underline{f}_{r-1} - Q \ , \ \ -p_i - \sum_{j'=1}^{j-1} \overline{g}_{i_{j'}} \ , \ \ d_i - \sum_{j'=1}^{j-1} \overline{g}_{i_{j'}} - Q(z_i^* - j)\}. \tag{A.7}$$

The first bound in equation (A.7) means that the vehicle cannot pick up from the location more product than its available space. Remember that a negative demand $g_r$ means that the location gives product to the vehicle. The second bound limits the total number of units collected at the location to those available in this location at this visit. The third bound takes into account the total demand in the rest of visits to the location (before and after the current visit).

Analogously, the upper bound of the demand interval can be set as

$$\overline{g}_r := \min\{\overline{f}_{r-1}\ ,\ \ q_i - p_i - \sum_{j'=1}^{j-1} \underline{g}_{i_{j'}}\ ,\ \ d_i - \sum_{j'=1}^{j-1} \underline{g}_{i_{j'}} + Q(z_i^* - j)\}. \tag{A.8}$$

Also, bounds for the load of the vehicle $f_r$ can be obtained from constraints (A.1) and (A.2). We set lower and upper bounds to the load interval as:

$$\underline{f}_r := \max\{0\ , \underline{f}_{r-1} - \overline{g}_r\} \tag{A.9}$$

$$\overline{f}_r := \min\{Q\ ,\ \ \overline{f}_{r-1} - \underline{g}_r\}. \tag{A.10}$$

When $r = 0$, we consider two variants of the problem. One variant fixes the initial load of the vehicle $f_0$ going out from the depot (e.g., $f_0 = 0$), and the other variant considers that the initial load of the vehicle can be any quantity. In the first case, we can fix $\underline{g}_0 = \overline{g}_0 = \underline{f}_0 = \overline{f}_0 = f_0$. In the second case, equations (A.7)–(A.10) can be applied considering $\underline{f}_{-1} = \overline{f}_{-1} = 0$.

*Appendix A.3. Readjusting demand and load intervals*

The bounds of a demand or load interval can change in some iteration of the algorithm. This change may lead to other readjustments in other intervals. We now describe six possible readjustments that are derived from the flow-conservation equations (A.2). Two readjustments correspond to the lower and upper bounds of the demand interval. Whereas, there are four posible readjustments for the load interval, depending on the bound (lower or upper), and if they are updated from the previous or later interval bounds. Note that a change of a load or demand bound can happen in an intermediate interval, not necessary the last one

that was set. Then, the six possible readjustments are:

$$\underline{g}_r := \underline{f}_{r-1} - \overline{f}_r \qquad \text{if } \underline{g}_r < \underline{f}_{r-1} - \overline{f}_r, \tag{A.11}$$

$$\overline{g}_r := \overline{f}_{r-1} - \underline{f}_r \qquad \text{if } \overline{g}_r > \overline{f}_{r-1} - \underline{f}_r, \tag{A.12}$$

$$\underline{f}_r := \underline{f}_{r-1} - \overline{g}_{r-1} \qquad \text{if } \underline{f}_r < \underline{f}_{r-1} - \overline{g}_{r-1}, \tag{A.13}$$

$$\overline{f}_r := \overline{f}_{r-1} - \underline{g}_{r-1} \qquad \text{if } \overline{f}_r > \overline{f}_{r-1} - \underline{g}_{r-1}, \tag{A.14}$$

$$\underline{f}_r := \underline{g}_{r+1} + \underline{f}_{r+1} \qquad \text{if } \underline{f}_r < \underline{g}_{r+1} + \underline{f}_{r+1}, \tag{A.15}$$

$$\overline{f}_r := \overline{g}_{r+1} + \overline{f}_{r+1} \qquad \text{if } \overline{f}_r > \overline{g}_{r+1} + \overline{f}_{r+1}. \tag{A.16}$$

Thus, if $\underline{f}_{r-1}$ or $\overline{f}_r$ have changed, the condition of equation (A.11) must be checked; if it is satisfied, $\underline{g}_r$ must be readjusted. For example, suppose that in a step of the algorithm $\underline{g}_r = -5$, $\underline{f}_{r-1} = 3$ and $\overline{f}_r = 8$. Remember that $\underline{g}_r = -5$ means that the vehicle must pick up at most 5 units of product in the visit $r$, $\underline{f}_{r-1} = 3$ means that the vehicle must load at least 3 units before visit $r$, and $\overline{f}_r = 8$ means that the vehicle must load at most 8 units after visit $r$ . Suppose that in a posterior step the value $\overline{f}_r$ changes to 7 (i.e., $\overline{f}_r = 7$). Then $\underline{g}_r < \underline{f}_{r-1} - \overline{f}_r$ and $\underline{g}_r$ is updated to $-4$ for equation (A.11).

Similarity, equations from (A.12) to (A.16) are used to readjust the demand and load intervals.

*Appendix A.4. Readjusting demand intervals of a location*

Given a location, the change in the bounds of a demand interval during a visit can lead to changes in other visits to this location. This is due to equations (A.3) and (A.4). Actually, this happens for the same reason as the second and third argument of the minimum and maximum functions of the equations (A.7) and (A.8).

More precisely, given a location $i$, let $j_c$ be the number of the visit such that $\overline{g}_{i_{j_c}}$ has changed, and let $j_s$ be the number of the last visit that has been set (by equations (A.7) and (A.8)). Then, for a visit $j$ to location $i$ satisfying $1 \leq j \leq j_s$ and $j \neq j_c$, we can readjust the lower bound of its demand interval to

$$\underline{g}_{i_j} := \max\{ -p_i - \sum_{\substack{1 \leq j' \leq j_s \\ j' \neq j}} \overline{g}_{i_{j'}} \ , \ \ d_i - \sum_{\substack{1 \leq j' \leq j_s \\ j' \neq j}} \overline{g}_{i_{j'}} - Q(z_i^* - j_s)\} \tag{A.17}$$

if the right hand side of this equation is greater than the current $\underline{g}_{i_j}$. Similarly, we can readjust the upper bound to

$$\overline{g}_{i_j} := \min\{q_i - p_i - \sum_{\substack{1\le j'\le j_s \\ j'\neq j}} \underline{g}_{i_{j'}}\ ,\ \ d_i - \sum_{\substack{1\le j'\le j_s \\ j'\neq j}} \underline{g}_{i_{j'}} + Q(z_i^* - j_s)\} \tag{A.18}$$

if the right hand side of this equation is less than current $\overline{g}_{i_j}$.

*Appendix A.5. Readjusting load intervals of a path*

We now present another procedure to readjust the load intervals. For a subset of consecutive visits (called path), the procedure calculates a lower bound and upper bound for the demand of this path, and checks whether these bounds are compatible with the load interval entering to this path and the load interval leaving this path.

Giving the circuit $C = (v_0, \ldots, v_k)$, a subset of consecutive visits of $C$ is denoted by $P = (v_{r_b}, \ldots, v_{r_e})$ with $0 < r_b < r_e \le k$. In addition, $\underline{g}_P$ and $\overline{g}_P$ denote a lower and an upper bound of the product demanded by path $P$, respectively. Then, we can readjust the load intervals using these four formulas:

$$\underline{f}_{r_e} := \underline{f}_{r_b-1} - \overline{g}_P \qquad \text{if } \underline{f}_{r_e} < \underline{f}_{r_b-1} - \overline{g}_P, \tag{A.19}$$

$$\overline{f}_{r_e} := \overline{f}_{r_b-1} - \underline{g}_P \qquad \text{if } \overline{f}_{r_e} > \overline{f}_{r_b-1} - \underline{g}_P, \tag{A.20}$$

$$\underline{f}_{r_b-1} := \underline{f}_{r_e} + \underline{g}_P \qquad \text{if } \underline{f}_{r_b-1} < \underline{f}_{r_e} + \underline{g}_P, \tag{A.21}$$

$$\overline{f}_{r_b-1} := \overline{f}_{r_e} + \overline{g}_P \qquad \text{if } \overline{f}_{r_b-1} > \overline{f}_{r_e} + \overline{g}_P. \tag{A.22}$$

They are derived from the flow-conservation constraints (A.2) and it is easy to see they limit the load of the vehicle entering or leaving $P$. For example, for equation (A.19), we suppose that $\underline{f}_{r_b-1} = 2$ and $\overline{g}_P = -3$. This means that the vehicle must carry at least 2 units of product entering $P$ and must collect at least 3 units of product when traveling through $P$. Therefore, the vehicle must leave $P$ with at least 5 units of product.

Next, we try to obtain tighten values for $\underline{g}_P$ and $\overline{g}_P$. We define $\underline{g}_P(i)$ and $\overline{g}_P(i)$ as the contribution of location $i$ to the bounds $\underline{g}_P$ and $\overline{g}_P$ such that $\underline{g}_P = \sum_{i\in I} \underline{g}_P(i)$ and $\overline{g}_P = \sum_{i\in I} \overline{g}_P(i)$. We fix values $\underline{g}_P(i)$ and $\overline{g}_P(i)$ depending on four cases: all visits to $i$ occur

inside $P$; no visit occurs inside $P$; first visit to $i$ occurs inside $P$ but last visit occurs outside $P$; and first visit to $i$ occurs outside $P$ but almost one visit to $i$ occurs inside $P$. For the first and second cases $\underline{g}_P(i) = \overline{g}_P(i) = d_i$ and $\underline{g}_P(i) = \overline{g}_P(i) = 0$, respectively . For the third and fourth cases, $\sum_{i_j \in P} \underline{g}_{i_j}$ and $d_i - \sum_{i_j \in C \setminus P} \overline{g}_{i_j}$ are lower bounds of $\underline{g}_P(i)$ and $\sum_{i_j \in P} \overline{g}_{i_j}$ and $d_i - \sum_{i_j \in C \setminus P} \underline{g}_{i_j}$ are upper bounds of $\overline{g}_P(i)$. In addition, for the third case, $-p_i$ is a lower bound $\underline{g}_P(i)$ and $q_i - p_i$ is an upper bound of $\overline{g}_P(i)$. Thus,

$$\underline{g}_P(i) := \begin{cases} d_i & \text{if } i_1, \ldots, i_{z_i^*} \in P, \\ 0 & \text{if } i_1, \ldots, i_{z_i^*} \in C \setminus P, \\ \max\{\sum_{i_j \in P} \underline{g}_{i_j}, d_i - \sum_{i_j \in C \setminus P} \overline{g}_{i_j}, -p_i\} & \text{if } i_1 \in P, i_{z_i^*} \in C \setminus P, \\ \max\{\sum_{i_j \in P} \underline{g}_{i_j}, d_i - \sum_{i_j \in C \setminus P} \overline{g}_{i_j}\} & \text{otherwise,} \end{cases}$$

and

$$\overline{g}_P(i) := \begin{cases} d_i & \text{if } i_1, \ldots, i_{z_i^*} \in P, \\ 0 & \text{if } i_1, \ldots, i_{z_i^*} \in C \setminus P, \\ \min\{\sum_{i_j \in P} \overline{g}_{i_j}, d_i - \sum_{i_j \in C \setminus P} \underline{g}_{i_j}, q_i - p_i\} & \text{if } i_1 \in P, i_{z_i^*} \in C \setminus P, \\ \min\{\sum_{i_j \in P} \overline{g}_{i_j}, d_i - \sum_{i_j \in C \setminus P} \underline{g}_{i_j}\} & \text{otherwise.} \end{cases}$$

*Appendix A.6. Main procedure*

The Algorithm 1 shows a pseudocode of the procedure CHECK_FEASIBILITY_CIRCUIT to evaluate if a circuit is feasible or not. In addition to the circuit $C = (v_1, \ldots, v_k)$, this procedure needs the parameters of a SD1PDTSP instance. They are the capacity $Q$, and parameters $d_i$, $p_i$ and $p'_i$ for each $i \in I$.

Some of the data structures of this procedure are the subsets $S_d$ and $S_f$, the variable *feasibility*, and the demand and load intervals for each position $r \in \{0, \ldots, k\}$. Subset $S_d$ stores the locations $i \in I$ where a demand interval has been modified. Subset $S_f$ stores the positions $r \in \{0, \ldots, k\}$ where a load interval has been modified. Variable *feasibility* stores one of the three status: true, false or undefined. At the beginning, *feasibility* is set to

**Algorithm 1** Procedure to check the feasibility of a circuit

**Require:** a circuit $C = (v_1, \ldots, v_k)$
**Ensure:** $C$ is feasible or infeasible

```
1: procedure CHECK_FEASIBILITY_CIRCUIT(C)
2:     S_d ← ∅                          ▷ locations i ∈ I whose a demand interval have been modified
3:     S_f ← ∅                ▷ positions r ∈ {0, ..., k} whose a load interval have been modified
4:     feasibility ← undefined                                  ▷ it can be true, false or undefined
                                ▷ All subroutines can access C, S_d, S_f, feasibility and intervals
5:     SET_DEMAND_AND_LOAD_INTERVALS(0)
6:     for r ← 1, k do                                  ▷ r is a counter for the position in the circuit
7:         SET_DEMAND_AND_LOAD_INTERVALS(r)
8:         READJUST_DEMAND_AND_LOAD_INTERVALS(r, r)
9:         if feasibility = false then
10:            return false
11:        end if
12:        while S_d ≠ ∅ or S_f ≠ ∅ do
13:            if S_d ≠ ∅ then
14:                Let i ∈ S_d
15:                S_d ← S_d \ {i}
16:                READJUST_DEMAND_INTERVALS_OF_A_LOCATION(i,r)
17:            else
18:                Let r_f ∈ S_f
19:                S_f ← S_f \ {r_f}
20:                READJUST_LOAD_INTERVALS_OF_PATHS(r_f,r)
21:            end if
22:            if feasibility = false then
23:                return false
24:            end if
25:        end while
26:    end for
27:    FIND_DEMANDS_AND_LOADS( )
28:    if feasibility = undefined then
29:        CHECK_THE_FEASIBILITY_OF_A_CIRCUIT_BY_MF_METHOD( )
30:    end if
31:    return feasibility
32: end procedure
```

undefined, and it is only updated if the feasibility or unfeasibility is certificated. We assume that all these data structures are accessible from any subroutine.

Subroutine SET_DEMAND_AND_LOAD_INTERVALS($r$) sets the demand and load intervals of a position $r$ of the circuit $C$ as it is described in Appendix A.2. It inserts location $v_r$ to $S_d$ and position $r$ to $S_f$.

Subroutine READJUST_DEMAND_AND_LOAD_INTERVALS($r', r$) readjusts the demand and load intervals of positions from $r'-1$ to 0 using equations (A.11), (A.12), (A.15) and (A.16). If neither the demand interval nor the load interval are updated for a given position, the procedure stops without having to reach position 0. In addition, if $r' < r$, it readjusts demand and load intervals of positions from $r'+1$ to $r$ using equations (A.11)–(A.14). Again, if neither the demand interval nor the load interval are updated of a given position, the procedure stops without having reach position $r$. If a demand and/or load interval are modified, subsets $S_d$ and $S_f$ are updated.

Subroutine READJUST_DEMAND_INTERVALS_OF_A_LOCATION($i, r$) readjusts the demands intervals of all visits to a location $i$ using equations (A.17) and (A.18). Obviously, if $z_i^* = 1$ (i.e., $i$ is visited only once) this subroutine does nothing. In addition, we find a little better performance if we call this subroutine only when the last visit to $i$ has happened. Inside of this subroutine, when a demand interval is modified, the procedure READJUST_DEMAND_AND_LOAD_INTERVALS is called.

Subroutine READJUST_LOAD_INTERVALS_OF_PATHS($r_l, r$) readjusts load intervals using equations (A.19)–(A.22). It considers subsets of visits $P = (v_{r_b}, \ldots, v_{r_e})$ such that $r_b \leq r_l \leq r_e$ and $r_b < r_e$. It is unnecessary to check all subsets satisfying these conditions. For example, if $v_{r_b}$ is visited only once and a condition of equations (A.19)–(A.22) is satisfied then the same condition for the path $P = (v_{r_b+1}, \ldots, v_{r_e})$ is satisfied also. Moreover, we find a little better performance if we check paths $P = (v_{r_b}, \ldots, v_{r_e})$ where $v_{r_b}$ corresponds to the first visit to a location visited more than once and $v_{r_e}$ corresponds to the last visit to a location visited more than once. Again, inside of this subroutine, when a load interval is modified, the procedure READJUST_DEMAND_AND_LOAD_INTERVALS is called.

Subroutine FIND_DEMANDS_AND_LOADS is a heuristic procedure to find values $g_r$ and

$f_r$ for $r = 0, \ldots, k$. It selects a position $r$ where $\underline{g}_r < \overline{g}_r$ and a value $g_r$ in the interval $[\underline{g}_r, \overline{g}_r]$, sets the bounds of the interval to this value $g_r$ a value (i.e., $\underline{g}_r := \overline{g}_r := g_r$), and calls the three subroutines above to readjust the rest of intervals. While there is a position $r$ where $\underline{g}_r < \overline{g}_r$, it repeats the process. If it finds a situation of infeasibility, the procedure restarts restoring the previous intervals and selecting other position or value $g_r$. Otherwise (when all intervals satisfy $\underline{g}_r = \overline{g}_r$ and the infeasibility condition was not reached), the procedure concludes that the circuit is feasible. Finally, if the loop is repeated more than a given number of times (2000 in our experiments) the procedure exits without determining feasibility.

When the FIND_DEMANDS_AND_LOADS fails to check the feasibility, we call the subroutine CHECK_THE_FEASIBILITY_OF_A_CIRCUIT_BY_MF_METHOD. This subroutine calls the method *MF* described in Section 5.2. Note that, this subroutine hardly ever is called (less than once each 1000 times) because of the feasibility or infeasibility of the circuit is certificated before.

*Appendix A.7. The non-preemptive case*

When preemption is forbidden, the main approach change only on how the bounds are calculated. Thus, equation (A.7) is replaced by:

$$\underline{g}_r := \begin{cases} \max\{0\ ,\ \ d_i - \sum_{j'=1}^{j-1} \overline{g}_{i_{j'}} - Q(z_i^* - j)\} & \text{if } d_i > 0 \\ \max\{\underline{f}_{r-1} - Q\ ,\ \ d_i - \sum_{j'=1}^{j-1} \overline{g}_{i_{j'}} - Q(z_i^* - j)\} & \text{if } d_i < 0 \end{cases} \tag{A.23}$$

and the equation (A.8) is replaced by:

$$\overline{g}_r := \begin{cases} \min\{\overline{f}_{r-1}\ ,\ \ d_i - \sum_{j'=1}^{j-1} \underline{g}_{i_{j'}} + Q(z_i^* - j)\} & \text{if } d_i > 0 \\ \min\{0\ ,\ \ d_i - \sum_{j'=1}^{j-1} \underline{g}_{i_{j'}} + Q(z_i^* - j)\} & \text{if } d_i < 0. \end{cases} \tag{A.24}$$

The rest of the equations remain the same except those in which the parameters $p_i$ or $q_i$ appear. In those cases, the terms of the minimum or maximum functions in which they appear should be removed from those equations.