# Decremental State-Space Relaxations for the Basic Traveling Salesman Problem with a Drone[*]

Marcos Blufstein[1] Gonzalo Lera-Romero[1] Francisco J. Soulignac[1,2]

[1]Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales. Departamento de Computación. Buenos Aires, Argentina.
[2]CONICET-Universidad de Buenos Aires. Instituto de Investigación en Ciencias de la Computación (ICC). Buenos Aires, Argentina.

**Abstract**

Truck-and-drone routing problems have become an important topic of research in the last decade due to their applications for last-mile deliveries. Despite the large number of publications in this area, the most efficient exact algorithms designed thus far struggle to solve the benchmark instances with 39 or more customers. This fact is true even for one of the simplest variants involving one truck and one drone whose routes must synchronize at customers' locations: the Basic Traveling Salesman Problem with a Drone (TSP-D). In this article, we devise a new algorithm for the TSP-D that is able to solve every benchmark instance with up to 59 customers, and it scales up to 99 customers when the drone is much faster than the truck. The core of our method is a dynamic programming algorithm that is applied for column generation and variable fixing within tailored decremental state-space relaxation strategies.

**Keywords:** traveling salesman problem with a drone, column generation, decremental state-space relaxation, dynamic programming, completion bounds, variable fixing.

## 1 Introduction

The use of unmanned aerial vehicles (a.k.a. drones) for delivering parcels has attracted a lot of attention in the last years, as reported in the numerous recent surveys on the subject (e.g. Chung et al., 2020; Liang and Luo, 2022; Macrina et al., 2020; Moshref-Javadi and Winkenbach, 2021; Otto et al., 2018). Drone-based logistic problems are usually classified according to whether the drones travel to nearby customers directly from depots or if they team up with supporting vehicles that carry them near the customers. In this article, we propose an exact algorithm to solve one of the simplest versions of a problem in the latter category, called the (basic) traveling salesman problem with a drone (TSP-D).

In the TSP-D, a truck teams up with a drone to visit a set of customers $V_c = \{1, \dots, n\}$. The truck starts its route at a depot 0, visits each customers from a subset $V_t$ of $V_c$ exactly once, and then it returns to the depot 0. Before leaving the depot, the truck is equipped with the drone which visits the remaining customers in $V_c - V_t$ exactly once. The truck carries the drone along its route while visiting the customers in $V_t$. Each time a vertex (customer or depot) $v \in V_t \cup \{0\}$ is reached, the drone can be launched from the truck to visit a customer $w$ in $V_c - V_t$, while the truck continues its journey. After visiting $w$, the drone travels to a customer (or depot) $z \neq v$ of $V_t$, where it is picked up by the truck to continue their journey together. Either vehicle can reach $z$ first, where it must wait for the other vehicle to arrive. The purpose of the TSP-D is to minimize the total time required by both vehicles to visit all the customers and return to the depot.

[*]Marcos Blufstein (mblufstein@dc.uba.ar), Gonzalo Lera-Romero (gleraromero@dc.uba.ar) Francisco J. Soulignac (fsoulign@dc.uba.ar)

The TSP-D, as described above, was proposed recently by Roberti and Ruthmair (2021). The reason why the problem is called "basic" is because it disregards several operational constraints that are commonly required in practice (see Section 1.1). In short, the TSP-D incorporates a main feature that is common to many problems in which a set of customers is visited by a truck and a drone: synchronization at customers' locations. In this sense, the TSP-D is analogous to the classical traveling salesman problem (TSP): whereas the TSP finds an optimal sequence of customers for a truck, the TSP-D generates an optimal synchronization of two sequences of customers visited by a truck and a drone. Synchronization is not exclusive to drone-based problems: see (e.g. Drexl, 2012; Gambella et al., 2018) for problems with alternative synchronization constraints.

Drone-based delivery problems were introduced in the seminal article by Murray and Chu (2015), who considered a problem with all the ingredients of the TSP-D, together with some additional operational constraints. Several generalizations of the TSP-D appeared since then, including, but no limited to, those in which: a single truck carries more than one drone (e.g. Cavani et al., 2021; Murray and Raj, 2020); several trucks work together, each taking care of its own set of drones (e.g. Poikonen et al., 2017; Wang et al., 2017); or a team of trucks carries a team of drones that can switch trucks (e.g. Bakir and Tiniç, 2020; Wang and Sheu, 2019). Some articles consider alternative scenarios where the drone may deliver more than one package per flight, the departure or landing of the drone may happen while the truck is en route, or the speed of the drone has to be adjusted (e.g. Masone et al., 2022; Poikonen and Golden, 2020).

Despite the advances on the operational aspects of the aforementioned problems, the exact algorithms designed for them struggle to solve even small-sized instances. Instead of considering a practical setting with different kinds of constraints, in this article we focus on the algorithmic aspects of the simplest TSP-D described above, aiming to push the computational limits as far as we can. Our main goal is to understand how to deal with the synchronization aspect, with the hope that the ideas we propose are instrumental for later developments. The algorithm we present builds upon the work by Roberti and Ruthmair (2021), who designed the most efficient exact algorithm for the TSP-D thus far. Even though we only consider the basic setting, most of the operational constraints described by Roberti and Ruthmair (2021) remain valid or can be incorporated with not much effort (though correctness proofs should be provided).

## 1.1 Brief review of the exact algorithms for the TSP-D

In this section we briefly review the exact algorithms proposed for different traveling salesman problems that involve a single truck and a single drone that delivers one package per flight. Moreover, we restrict ourselves to those cases in which the drone leaves and meets the truck at customers' locations. We refer to the surveys in Section 1 for a broader view that includes variants of these problems and/or references to the heuristic solutions proposed thus far.

The starting point for truck-and-drone delivery problems is the Flying Sidekick TSP (FSTSP) introduced by Murray and Chu (2015). The FSTSP is one of the most studied generalizations of the TSP-D, and it incorporates the following constraints on top of the TSP-D:

$c_1$: some customers cannot be visited by the drone,

$c_2$: the time required for launching and landing the drone cannot be depreciated,

$c_3$: the battery of the drone has a limited endurance that bounds its airborne time, and

$c_4$: the drone cannot save battery by landing at a rendezvous customer before the truck arrives.

The other most studied generalization of the TSP-D is the TSP-D with loops and revisits (TSP-D+lr) introduced by Agatz et al. (2018). (Agatz et al. 2018 refer to their problem simply as the TSP-D. We reserve this acronym for the basic TSP-D that we study in this paper, thus the non-standard TSP-D+lr acronym.) The TSP-D+lr incorporates $c_1$ and the following features:

$c_5$: the truck may revisit some customers,

$c_6$: the truck need not visit a customer while the drone is airborne; that is, the drone may leave the truck at a vertex $v$, visit a customer $w$, and then return to $v$, while the truck remains stationary at $v$.

Less popular variants of the TSP-D were studied alongside the FSTSP and the TSP-D+lr. These variants introduce alternative objective functions (e.g. Ha et al., 2018), non-straight drone routes that avoid no-fly zones (e.g. Jeong et al., 2019), and additional constraints such as:

$c_7$: a weight-dependent battery consumption for the drone (Jeong et al., 2019), and

$c_8$: a limit on the number of customers that the truck can visit while the drone is airborne (Bouman et al., 2018).

The TSP-D, as studied in this article, was introduced by Roberti and Ruthmair (2021). To justify their research on this basic problem, the authors state that:

> “the timely synchronization of vehicles has already been identified as a particularly difficult feature to handle in routing problems (see, e.g., Drexl, 2012). [...We] believe that both from a theoretical and practical point of view it is important to investigate simple truck-and-drone problems with the prospect of applying gained knowledge and extending efficient solution approaches to more complicated problem variants.[...] In the light of the existing work, [...it is important to] focus on a basic variant of the TSP-D that is a special case of many problems addressed in the literature because methodological contributions on this basic TSP-D can be instrumental to solve more general cases.”

To strengthen their arguments, Roberti and Ruthmair (2021) show how constraints $c_1$–$c_4$ and $c_6$–$c_7$ can be incorporated into their algorithms without much effort or impact on efficiency. Dell’Amico et al. (2021a) provide a similar categorization of TSP-D variants according to whether different components (constraints) are activated or not.

Table 1 summarizes the contributions regarding exact methods for solving different variants of the TSP-D. The focus is on the number of customers that each algorithm can handle efficiently, according to the computational experiments carried in the respective article. To be fair, we exclude those settings that aggressively reduce the search space with extra constraints such as $c_8$. Column Time Limit indicates the time limit imposed for the experiments, whereas columns All, 50%, and Max show the maximum number of customers for which the algorithm could solve all, at least half, or at least one of the instances evaluated, respectively. There is some room for interpretation here, as not all the information is readily available in every publication; we use blank entries when we cannot find some value and question marks where the information is not clear.

Table 1 is not meant to compare the reported algorithms, as different variants and configurations are summarized. Still, Table 1 sends a clear message: the exact algorithms proposed thus far struggle to solve instances with 30 or more customers, regardless of which variant of the TSP-D is solved. Table 1 also highlights that different techniques were applied to solve TSP-D problems. Earliest approaches focused on modeling and simpler B&B and DP algorithms that could reach solutions with up to 15 customers. The newest methods use more elaborated techniques to reach 30 customers. These techniques include: branch-and-cut with adapted cutting planes from the TSP (van Dijck and Bouman, 2018), branch-and-price with state-of-the-art relaxations for the pricing problem that is solved with dynamic programming (Roberti and Ruthmair, 2021), branch-and-cut with tailored column-and-row generation procedures (Boccia et al., 2021b), and Benders decomposition (Vásquez et al., 2021). Also, some effort was devoted to devise efficient MILP formulations that are compact and avoid big-$M$ constraints (Boccia et al., 2021a; Dell’Amico et al., 2022).

## 1.2 Our Contributions

In this article, we combine two decremental state-space relaxation strategies (DSSR) to solve the TSP-D. Our main solver applies column generation to compute an optimal route $r$ within a relaxed state-space of routes $\Omega$. Then, it calls a secondary solver that applies a DSSR to find an optimal route $r'$ in a restricted

| Publication | Technique | Time Limit | All | 50% | Max |
|---|---|---|---|---|---|
| Murray and Chu (2015) | MILP | 0.5hs | 0 | 0 | 0 |
| Ponza (2016) | MILP | 24hs? | — | 9 | 9 |
| Agatz et al. (2018) | MILP | ≈2hs? | — | 11? | 11 |
| Bouman et al. (2018) | DP | 12hs | — | 15 | 15 |
| Es Yurek and Ozmutlu (2018) | Iter(MILP) | 1h | 12 | 12 | 12 |
| Ha et al. (2018) | MILP | 1h? | — | 9? | 9 |
| van Dijck and Bouman (2018) | B&C | 1h? | — | 19* | 19* |
| Jeong et al. (2019) | MILP | 12hs? | — | 9? | 9 |
| Poikonen et al. (2019) | B&B | 1h | — | 9 | 9 |
| Tang et al. (2019) | CP | 1h? | — | 17? | 17 |
| Schermer et al. (2020) | B&C | 1h | 10 | 19 | 20 |
| Boccia et al. (2021b) | B&C(C&RG) | 1h | 10 | 20 | 20 |
| Boccia et al. (2021a) | B&C | 2hs | 15 | 20 | 20 |
| Dell'Amico et al. (2021b) | B&C | 1h | — | 9 | 13 |
| Dell'Amico et al. (2021) | B&B | 1h | 14 | 14 | 19 |
| Roberti and Ruthmair (2021) | B&P(DP) | 1h | 19 | 29 | 39 |
| Vásquez et al. (2021) | Benders | 3hs | 19 | 22 | 25 |
| Dell'Amico et al. (2022) | MILP,B&C | 1h | 10 | 20 | 20 |
| **This paper** | DSSR(CG,DP) | 1h | 59 | 79 | 99 |

Table 1: Overview of the different exact algorithms for variants of the TSP-D. The algorithm by van Dijck and Bouman (2018) limits to four the number of customers that the truck can visit while the drone is flying. The acronyms in the second column refer to: mixed integer linear programming formulation (MILP), iterative procedure that solves a MILP (Iter(MILP)), dynamic programming (DP), branch-and-bound (B&B), branch and cut (B&C), constraint programming (CP), B&C with column-and-row generation (B&C(C&RG)), branch-and-price with a DP for pricing (B&P(DP)), Benders decomposition (Benders), and decremental state-space relaxation with column generation and a DP for pricing (DSSR(CG,DP)).

state-space $\Omega' \subseteq \Omega - \{r\}$. If $r'$ is a feasible solution to the TSP-D, then $r'$ is an optimum solution, thus the main solver stops; otherwise, the main solver continues its search in a restricted state-space included in $\Omega - \{r\}$.

Relaxed state-spaces were introduced by Christofides et al. (1981), for computing lower bounds to routing problems using DP algorithms, whereas the first DSSRs were proposed by Boland et al. (2006) and Righini and Salani (2008). Within truck-only routing problems, one of the most efficient state-space relaxation is the *ng*-route relaxation devised by Baldacci et al. (2011, 2012). The notion of *ng*-feasibility was adapted to the TSP-D by Roberti and Ruthmair (2021), who proposed a tailored DP algorithm to compute an optimal *ng*-feasible route.

To contribute to the TSP-D, in this work:

1. We provide a new state-space relaxation, through a stronger notion of *ng*-feasibility, that yields greater lower bounds and has a limited impact on the time required to solve each relaxation.

2. We describe an improved DP algorithm to compute an optimal *ng*-feasible route, that applies stronger dominance rules based on the concept of *partial* dominance (see, e.g. Lera-Romero et al., 2020, 2022).

3. We propose a DSSR for a solver that, given a route $r$ and $ub \in \mathbb{R}$, decides if $ub$ is an upper bound of the optimum value. The solver repeatedly runs the DP algorithm, each time on a more restricted state-space, while it applies a variable fixing method based on completion bounds. Even though this is a general solver for the TSP-D, we use it as a secondary method because its efficiency depends on how close are $ub$ and a lower bound implied by $r$.

4. We introduce a tailored column generation algorithm that solves the linear relaxation of the master problem faster than a classical column generation algorithm, while it also provides tighter lower bounds for the optimum value of the master problem. The algorithm works by solving easier pricing problems that are obtained by preemptively fixing some variables according to completion bounds.

5. Finally, we devise a main solver that combines the tailored column generation algorithm and the secondary solver within a DSSR to solve the TSP-D.

To reduce the state-space, both solvers apply a neighborhood augmentation method that adapts to the TSP-D the ideas proposed by Tilk and Irnich (2017).

The article is organized as follows. In Section 2, the TSP-D is formally stated together with the new notion of *ng*-feasibility that defines the state-space relaxation. In Section 3, the DP algorithm that solves the relaxed TSP-D is described and new completion bounds for partial routes are obtained. The neighborhood augmentation algorithm that we use to restrict the state-space is described in Section 4. The secondary solver, called iterative fixing, is described in Section 5, together with other variable fixing methods that apply completion bounds. Section 6 presents the tailored column generation method. Column generation, iterative fixing, and neighborhood augmentation are combined in Section 7 to create the main solver for the TSP-D. Section 8 evaluates the efficiency of the main solver while it tests the impact of each of its components in an isolated environment. Finally, future research directions are provided in Section 9.

## 2 Problem Statement

Throughout the article we use $n$ to denote the number of customers. To describe the transport network, we use a complete digraph $D = (V, A)$ with vertex set $V = \{0, \ldots, n\}$. Vertex 0 represents the depot, $V_c = \{1, \ldots, n\}$ represents the set of customers, and each arc $ij \in A$ represents the direct route from $i$ to $j$. The arc $ij$ has three positive weights: $c^{\mathrm{T}}_D(i, j)$ is the time required by the truck to travel from $i$ to $j$; $c^{\mathrm{F}}_D(i, j)$ is the time spent by the drone when it leaves the truck at $i$ to visit $j$ alone; and $c^{\mathrm{J}}_D(i, j)$ is the time consumed by the drone to travel from $i$ to meet the truck again at $j$. Any of these weights can be infinite, meaning that the direct route is not available.

Following Roberti and Ruthmair (2021), we find it convenient to describe routes as sequences of moves (Figure 1). With this purpose in mind, we say a *partial route* is a sequence $p = (v_1, x_1), \ldots, (v_k, x_k)$, where $v_i$ is a vertex and $x_i \in \{\mathrm{d}, \mathrm{t}\}$ for each $1 \leq i \leq k$, and $x_1 = \mathrm{d}$. Each pair $(v_i, x_i)$ is a *move* of $p$. Intuitively, $v_i$ is the $i$-th vertex that is visited by at least one vehicle, while $x_i = \mathrm{d}$ if the drone visits $v_i$. Formally, the truck and the drone visit different customers and end at the locations $t(p)$ and $d(p)$, respectively, that are recursively computed as follows. If $p = (v_1, \mathrm{d})$, then $t(p) = d(p) = v_1$ to indicate that the route *starts* at $v_1$. If $p = q + (v_k, x_k)$, then $v_k \neq t(q)$ and there are three possibilities: if $x_k = \mathrm{t}$, then the truck moves from $t(q)$ to $t(p) = v_k$ and the drone stays at $d(p) = d(q)$; if $x_k = x_{k-1} = \mathrm{d}$, then the drone is inside the truck at location $t(q) = d(q)$, hence the truck carries the drone to the location $t(p) = d(p) = v_k$; finally, if $x_{k-1} = \mathrm{t}$ and $x = \mathrm{d}_k$, then the drone flies from $d(q)$ to visit $v_k$ and then it flies to meet the truck at $d(p) = t(p) = t(q)$.

For the partial route $p$, the position $b(p) = \max\{j \mid x_j = \mathrm{d} \text{ for } 1 \leq j \leq k\}$ is referred to as the *bifurcation* point of $p$ (Figure 1). Clearly, the truck and the drone end at the same location of the partial route $(v_1, x_1), \ldots, (v_{b(p)}, x_{b(p)})$, whereas the moves $(v_{b(p)+1}, \mathrm{t}), \ldots, (v_k, \mathrm{t})$ describe the route taken by the truck after the bifurcation point. In the real-world operation, the drone immediately takes off from the truck at the bifurcation vertex to visit a yet unspecified vertex $v_{k+j}$, for $j > 0$. As $v_{k+j}$ is unknown, it is impossible to know which vehicle will wait at the meeting point $v_{k+j-1}$. Thus, the cost of $p$ is specified by two values $c_D(p)$ and $\tau_D(p)$ that denote the time required by both vehicles to reach $v_{b(p)}$ and the time consumed by the truck to reach $v_k$ from $v_{b(p)}$, respectively. Clearly, $c_D((v_1, \mathrm{d})) = 0$ and $\tau_D(p) = \sum_{i=b(p)}^{k-1} c^{\mathrm{T}}(v_i, v_{i+1})$, while $c_D(p)$ can be recursively computed as follows when $p = q + (v_k, x_k)$ for a partial route $q$:

$$c_D(p) = \begin{cases} c_D(q) & \text{if } x_k = \mathrm{t} \\ c_D(q) + c^{\mathrm{T}}(t(q), v_k) & \text{if } x_k = x_{k-1} = \mathrm{d} \\ c_D(q) + \max\{\tau(q), c^{\mathrm{F}}(d(q), v_k) + c^{\mathrm{J}}(v_k, t(q))\} & \text{otherwise} \end{cases}$$

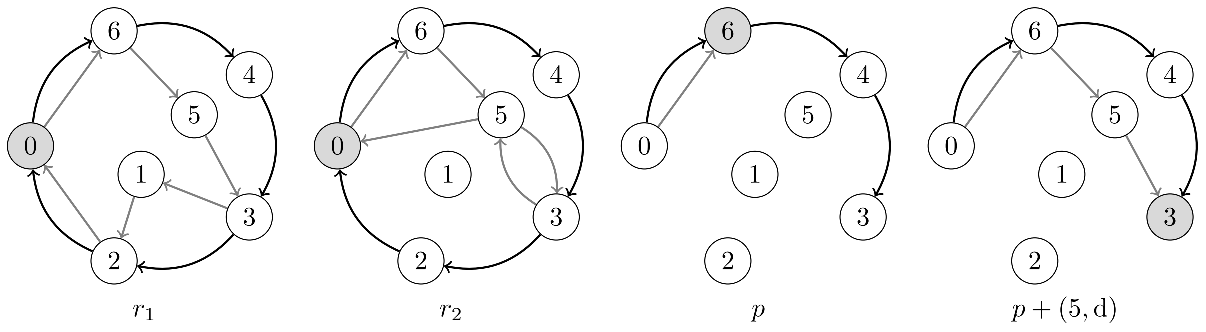

Figure 1: Graphical representation of the partial routes $r_1 = (0, \mathrm{d})$, $(6, \mathrm{d})$, $(4, \mathrm{t})$, $(3, \mathrm{t})$, $(5, \mathrm{d})$, $(2, \mathrm{t})$, $(1, \mathrm{d})$, $(0, \mathrm{d})$, $r_2 = (0, \mathrm{d})$, $(6, \mathrm{d})$, $(4, \mathrm{t})$, $(3, \mathrm{t})$, $(5, \mathrm{d})$, $(2, \mathrm{t})$, $(0, \mathrm{t})$, $(5, \mathrm{d})$; $p = (0, \mathrm{d})$, $(6, \mathrm{d})$, $(4, \mathrm{t})$, $(3, \mathrm{t})$; and $p + (5, \mathrm{d})$ on a network with 6 customers. The truck and drone traverse the black and gray arcs, respectively, thus the truck carries the drone from $v$ to $w$ when arcs of both colors join $v$ to $w$. The gray vertex marks the position of the drone. Partial routes $r_1$ and $r_2$ are routes, as they perform eight moves and visit six customers. Only $r_1$ is elementary, as it visits all the customers exactly once. Note that the drone always moves last in a route, either carried by the truck as in $r_1$ or alone as in $r_2$. In $r_1$ the truck also moved last (to carry the drone), whereas in $p_3$ the truck alone moved last. The bifurcation point of $p$ is $b(p) = 2$, thus $c(p) = c^{\mathrm{T}}(0, 6)$ and $\tau(p) = c^{\mathrm{T}}(6, 4) + c^{\mathrm{T}}(4, 3)$. As the drone alone moves last in $p + (5, \mathrm{d})$, it follows that $c(p + (5, \mathrm{d})) = c^{\mathrm{T}}(0, 6) + \max\{c^{\mathrm{T}}(6 + 4) + c^{\mathrm{T}}(4, 3), c^{\mathrm{F}}(6, 5) + c^{\mathrm{J}}(5, 3)\}$ and $\tau(p + (5, \mathrm{d})) = 0$.

As usual, $D$ is omitted from $c_D$ when no confusions are possible. From now on, we say that the *drone is inside the truck* in $p$ if $b(p) = k$ (i.e., $x_k = \mathrm{d}$), that the *truck moved last* in $p$ if $x_k = \mathrm{t}$ or the drone is inside the truck in $q$, that the *drone moved last* in $p$ if $x_k = \mathrm{d}$, and that the *truck carried the drone in the last move of* $p$ if both vehicles moved last in $p$. If only one vehicle moved last, then that vehicle moved *alone*.

We say that a partial route $r$ is a *route* when it starts at the depot 0, visits $n = |r| - 2$ customers, and the truck and drone end at the depot 0; Figure 1. (Observe that only one move other than the first involves the depot 0; this move is either the last move $(0, \mathrm{d})$ or the previous to last move $(0, \mathrm{t})$.) By definition, a route $r$ can visit the same customer more than once, in which case some customers are not visited by $r$. Those partial routes that visit each customer exactly once are called *elementary*. Clearly, $r$ is elementary when the customers of its moves are all different.

The TSP-D is the problem of finding an elementary route of minimum cost when a digraph $D$ is given. The TSP-D can be modeled as a set covering problem, observing that it suffices to find an optimal set of routes $\mathcal{R}$ of $D$ that together visit all the customers, where $\mathcal{R}$ has at most one route. Let $\Omega$ be a family of routes of $D$ that contains every elementary route and $a_{vr}$ be the number of times a route $r \in \Omega$ visits a vertex $v \in V$. Then, for decision variables $y_r$ that indicate if $r \in \Omega$ belongs to the solution set $\mathcal{R}$, the TSP-D is solved by the integer program

$$\mathrm{MP}(D, \Omega)\colon c^*(D, \Omega) = \min \sum_{r \in \Omega} c(r) y_r \tag{1}$$

$$\text{s.t.} \sum_{r \in \Omega} y_r \leq 1 \tag{2}$$

$$\sum_{r \in \Omega} a_{vr} y_r \geq 1 \qquad \text{for every } v \in V_c \tag{3}$$

$$y_r \in \{0, 1\} \qquad \text{for every } r \in \Omega. \tag{4}$$

The objective function (1) minimizes the total costs of the routes in the solution $\mathcal{R}$, constraint (2) enforces that at most one route is selected for $\mathcal{R}$, constraints (3) guarantee that every customer is visited, and (4) impose the domain of the $y$ variables.

To solve the TSP-D, we compute a sequence of lower bounds $lb_0 < \ldots < lb_i = c^*$ of the optimum value $c^*$, using a DSSR that combines neighborhood augmentation (Section 4), variable fixing (Section 5) and column generation (Section 6). What these methods have in common is that they depend on solving different pricing problems. Given $D$, $\Omega$, and a vector $u \in \mathbb{R}^{n+1}$ such that $u_0 \leq 0$ and $u_v \geq 0$, $v \in V_c$, the *pricing problem* is to find a family of routes $\mathcal{R} \in \Omega$ that contains at least one route $r$ minimizing

$$\bar{c}(D, u, r) = c(r) - u_0 - \sum_{v \in V_c} a_{vr} u_v.$$

We call $u$ a *dual vector* and we say that $\bar{c}(D, u, r)$ is the *$u$-cost* of $r \in \Omega$. In general, the *$u$-cost* of a partial route $p = (v_1, x_1), \ldots, (v_k, x_k)$, $k < n+2$, is $\bar{c}(D, u, p) = c(p) - \sum_{i=1}^{k} u_{v_i}$. Certainly, the $u$-cost of $r \in \Omega$ is the reduced cost of $r$ when $u$ is a dual solution of the linear relaxation of $\mathrm{MP}(D, \Omega)$.

Each dual vector $u$ defines a lower bound of the optimum value $c^*$, because

$$\overline{lb}(D, \Omega, u) = \min\{\bar{c}(D, u, r) \mid r \in \Omega\} + \sum_{v \in V} u_v \leq \bar{c}(D, u, r^*) + \sum_{v \in V} u_v = c(r^*),$$

for every elementary route $r^*$. We refer to $\overline{lb}(D, \Omega, u)$ as being the *$u$-bound* of the instance $(D, \Omega)$. From now on, we omit $D$, $\Omega$, and $u$ from MP, $c^*$, $\bar{c}$, and $\overline{lb}$ when they are clear by context.

## 2.1 The *ng*-route relaxation

The definition of $\Omega$ has a major impact on the quality of the $u$-bound $\overline{lb}$ that is obtained after solving a pricing problem on a dual vector $u$, as well as on the time required to solve it. If, on the one hand, $\Omega$ is taken as the family of elementary routes, then $\overline{lb} = c^*$ is as tight as possible. However, each pricing problem is strongly NP-hard, as its optimum is precisely the optimum of the TSP-D. If, on the other hand, $\Omega$ is relaxed to the family of all routes, then the pricing problem can be solved in pseudo-polynomial time. This time, however, $\overline{lb}$ is usually too far from $c^*$. A better approach is to limit the relaxation of $\Omega$ to obtain a tight lower bound with a not-so-hard pricing problem.

The *ng*-route relaxation is one the most efficient state-space relaxations proposed in the literature for routing problems without drones (Baldacci et al., 2011). The idea is to define, for each customer $v$, a *neighborhood* $N(v)$ that contains customers that are "close" to $v$. Then, a (partial) path $p$ is *ng-feasible* if for each of its cycles $v_1, \ldots, v_j, v_1$ it happens that $v_1 \notin N(v_i)$, for some $2 \leq i \leq j$. In the *ng-route relaxation*, $\Omega$ is the set of *ng*-feasible routes. The quality of the lower bound and the efficiency of the pricing problem depend on the size of the neighborhoods; larger neighborhoods yield tighter bounds, whereas smaller neighborhoods imply faster algorithms.

Due to the simultaneous operation of the two vehicles, the *ng*-route relaxation can be generalized to the context of the TSP-D in many different ways. The definition given by Roberti and Ruthmair (2021) depends on a set $F^{\mathrm{T}}(p)$ that denotes the customers that cannot be visited in the next move of $p$. If $|p| = 1$, then $F^{\mathrm{T}}(p) = \emptyset$, while if $p = q + (v, x)$ for a partial route $q$, then:

$$F^{\mathrm{T}}(p) = \begin{cases} (F^{\mathrm{T}}(q) \cup \{t(q)\}) \cap N(v) & \text{if the truck moved in } p \\ (F^{\mathrm{T}}(q) \cup \{v\}) \cap N(t(q)) & \text{otherwise} \end{cases}$$

With the above definition, Roberti and Ruthmair (2021) declare a partial route $p$ as *ng*-feasible when either $|p| = 1$ or $p = q + (v, x)$ for an *ng*-feasible partial route $q$ such that $v \notin F^{\mathrm{T}}(q) \cup \{t(q), d(q)\}$.

It can be easily seen that the path visited by the truck in an *ng*-feasible partial route is always *ng*-feasible as defined for the problems without drones, because the neighborhood $N(v)$ is ignored for each move $(v, \mathrm{d})$ in which the drone visits $v$ alone. On the contrary, from its myopic view, the drone may visit the same customer twice without leaving its neighborhood. For instance, Figure 2 depicts two partial routes $p_1$ and $q$ in which the drone sequentially visits $w$, $v$ and $w$, even though $w \in N(v)$. This does not contradict the definition of *ng*-feasibility because $w \notin N(z)$ for a move $(z, \mathrm{t})$ that lies between both moves involving $w$. This situation is quite common in the TSP-D, as the drone is encouraged to visit near vertices to avoid

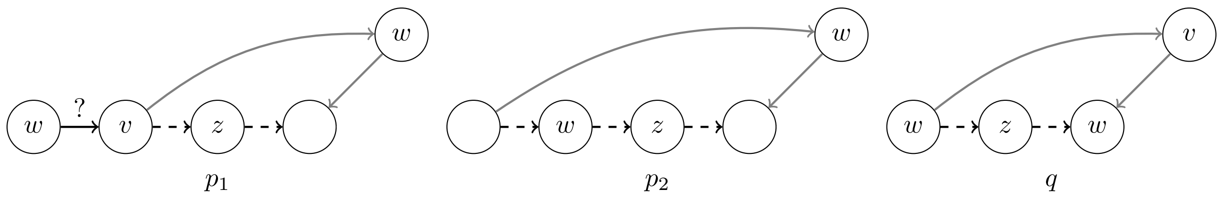

Figure 2: Examples of partial routes that are *ng*-feasible according to the definition by Roberti and Ruthmair (2021), some of which are not *ng*-feasible according to Definition 1. Arcs are colored as in Figure 1, a question mark over an arc means that either or both vehicles can move, and dashed arcs represent paths. The vertices appear from left to right in the same order as their corresponding moves in each partial route; the vertices on the top row are visited by the drone alone.

waiting times. A different source of repetitions arises from the fork of the truck and drone routes. Figure 2 exhibits a partial route $p_2$ in which both the truck and the drone visit the same vertex $w$ after the truck and drone routes fork and before they join again. The route is *ng*-feasible because $w \notin N(z)$. Once again, this situation is quite common in the TSP-D, where both the drone and the truck move to near vertices.

We propose a stronger version of *ng*-feasibility in which the customers that the drone can visit alone are further restricted. For a partial route $p$, a set $F^{\mathrm{D}}(p)$ containing the customers that the drone cannot visit before it meets the truck again is recursively defined as follows:

$$F^{\mathrm{D}}(p) = \begin{cases} F^{\mathrm{T}}(p) \cup \{d(p)\} & \text{if the drone is inside the truck in } p \\ F^{\mathrm{D}}(q) \cup \{v\} & \text{otherwise (i.e., if } p = q + (v, \mathrm{t})) \end{cases}$$

The notion of *ng*-feasibility used in this article is recursively defined as follows.

**Definition 1** (*ng*-feasibility)**.** *Every partial route $p$ with $|p| = 1$ is ng-*feasible*. A partial route $p = q + (v, x)$ is ng-*feasible*, for a move $(v, x)$, when $q$ is ng-feasible, $v \notin F^{\mathrm{T}}(q) \cup \{t(q)\}$ if the truck moved in $p$, and $v \notin F^{\mathrm{D}}(q)$ otherwise (i.e., the drone moved alone in $p$).*

Intuitively, while the truck carries the drone, $F^{\mathrm{T}}(p)$ keeps the customers that cannot be visited in the next move because of the *ng*-route relaxation. Instead, $F^{\mathrm{D}}(p)$ precomputes the customers that the drone cannot visit if the truck moves alone. Then, after the bifurcation point, $F^{\mathrm{D}}(p)$ simply records each customer visited by the truck to enforce a visit to a different customer by the drone. Thus, under this new rule for the relaxation, the partial routes $p_1$ and $p_2$ in Figure 2 are not *ng*-feasible. (We remark that the path taken by the drone may still visit the same vertex without leaving its neighborhood, as in the partial route $q$ of Figure 2.) Although the new *ng*-route relaxation provides tighter bounds, it also implies solving harder pricing problems. The computational experiments in Section 8 show that the trade-off of using a stronger rule is convenient in practice.

An important difference between the notion of *ng*-feasibility given by Roberti and Ruthmair (2021) and Definition 1 is that the latter does not require $v \neq d(q)$ when the truck visits a customer $v$ alone. Hence, the latter *ng*-feasibility allows the truck and the drone to be at the same location $t(p) = d(p)$, even if the drone is outside the truck. This decision allows us to implement a stronger dominance rule and, furthermore, it simplifies the implementation of a bidirectional search. In short, Rule 3 and Theorem 1 are false if $v$ is required to be different than $d(p)$.

## 3 An Algorithm for the Pricing Problem

Recall that the goal of the pricing problem is to find a route $r$ with minimum $u$-cost $\bar{c}(r) = c(r) - u_0 - \sum_{v \in V_c} a_{vr} u_v$ when a dual vector $u$ is given as input, where $a_{vr}$ is the number of times $r$ visits $v \in V_c$. We solve this problem with a labeling algorithm that enhances the one proposed by Roberti and Ruthmair (2021) with the stronger *ng*-feasibility and with new, stronger, dominance rules.

Our *labeling algorithm* works by exploring a tree whose nodes are *ng*-feasible partial routes and whose leaves are *ng*-feasible routes. The partial route $(0, \mathrm{d})$ is the root, and $p$ is the parent of each partial route $p + (v, x)$. Moreover, the location of the truck and drone is the depot $t(p) = d(p) = 0$ when $|p| = n + 2$. Thus, the last move always takes the drone from $d(p) \neq 0$ to the depot 0, while the truck either carries the drone from $t(p) = d(p)$ to 0 or waits at the depot. Starting with a tree that contains only the root node $(0, \mathrm{d})$, the labeling algorithm repeatedly *extends* a leaf node $p$ with $|p| < n + 2$ into a child node $p + (v, x)$, for every move $(v, x)$ such that $p + (v, x)$ is *ng*-feasible and $\bar{c}(p + (v, x)) < \infty$. The algorithm by Roberti and Ruthmair (2021) works analogously, although it extends partial routes that are *ng*-feasible under their weaker definition. Unless otherwise stated, from now on we assume that every partial route starts at 0.

To improve the efficiency of the labeling algorithm, dominance rules are applied to discard partial routes, without extending their subtrees. A simple notion of dominance is exemplified by the next definition.

**Definition 2.** *An ng-feasible partial route $p$* dominates *an ng-feasible partial route $q$ when $p + q'$ is an ng-feasible route with $\bar{c}(p + q') \leq \bar{c}(q + q')$, for every ng-feasible route $q + q'$.*

Clearly, avoiding the extension of any dominated partial route $q$ does not affect the correctness of the algorithm, as far as at least one partial route $p$ dominating $q$ is extended, because at least one route with a $u$-cost not greater than the discarded routes always survives. Roberti and Ruthmair (2021) propose a dominance rule to discard partial routes within their algorithm. To provide a point of comparison only, below we adapt this dominance rule to deal with our stronger notion of *ng*-feasibility: condition (b) is new and reflects how much harder is to solve the pricing problem. Hereafter, we let $\bar{\tau}(p) = \bar{c}(p) + \tau(p)$ be the $u$-cost that we already know that $p$ has to pay, for every partial route $p$.

**Dominance Rule 1** (Roberti and Ruthmair (2021))**.** *Let $p$ and $q$ be ng-feasible partial routes such that the drone is inside the truck in either both or none of $p$ and $q$. If (a) $|p| = |q|$, $t(p) = t(q)$, $d(p) = d(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $\bar{\tau}(p) \leq \bar{\tau}(q)$, (b) $F^{\mathrm{D}}(p) \subseteq F^{\mathrm{D}}(q)$, and (c) either:*

1. *$\bar{c}(p) \leq \bar{c}(q)$, or*
2. *$\bar{c}(p) + c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, t(p)) \leq \bar{\tau}(q)$ for every $w \in V_c$,*

*then $p$ dominates $q$.*

As defined above, dominance is an all or nothing relation between $p$ and $q$: the extensions to all the children of $q$ are forbidden if $p$ dominates $q$, and all of them are allowed otherwise. Partial dominance is a generalization of the dominance relation that forbids a subset of extensions even when $q$ is not dominated by $p$ (see e.g. Lera-Romero et al., 2020, 2022). In this article, we implement the following notions of partial dominance (Figure 3).

**Definition 3.** *Suppose $p$ and $q$ are ng-feasible partial routes in which the drone is outside the truck, and let $w \in V_c$. We say that $p$* dominates $q$ via $w$ *when $r_p = p + q_1 + q_2$ is an ng-feasible route with $\bar{c}(r_p) \leq \bar{c}(r_q)$, for every ng-feasible route $r_q = q + q_1 + q_2$ such that $q_1 = (v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d})$.*

**Definition 4.** *Suppose $p$ and $q$ are ng-feasible partial routes in which the drone is outside the truck, and let $v, w \in V_c$. We say that $p$* forking to $v$ dominates $q$ via $w$ *when $r'_p = p + (v, \mathrm{d}), (v_1, \mathrm{d}), \ldots, (v_k, \mathrm{d}) + q_2$ is an ng-feasible route with $\bar{c}(r'_p) \leq \bar{c}(r_q)$, for every ng-feasible route $r_q = q + (v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d}) + q_2$.*

As done for dominated partial routes, we can discard the extensions of those partial routes of the form $q + (v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d})$, as far as some partial route $p$ dominating $q$ via $w$ is extended. Thus, even if we cannot forbid all the extensions of $q$, we can still avoid the exploration of large part of the labeling tree. Regarding the implementation of dominance via $w$, recall that $F^{\mathrm{D}}(q)$ denotes the set of customers that the drone cannot visit until it meets the truck again. Therefore, it suffices to insert $w$ into $F^{\mathrm{D}}(q)$ when $q$ is detected to be dominated via $w$. However, to avoid giving two similar but non-equal meanings to $F^{\mathrm{D}}$, we write $F^{\mathrm{D}}_{\mathrm{lbl}}(p)$ to denote the set of customers that the drone cannot visit next within the labeling algorithm. Observe that $F^{\mathrm{D}}_{\mathrm{lbl}}(p) = F^{\mathrm{D}}(p)$ when the drone is inside the truck in $p$, while $q$ can be discarded the moment

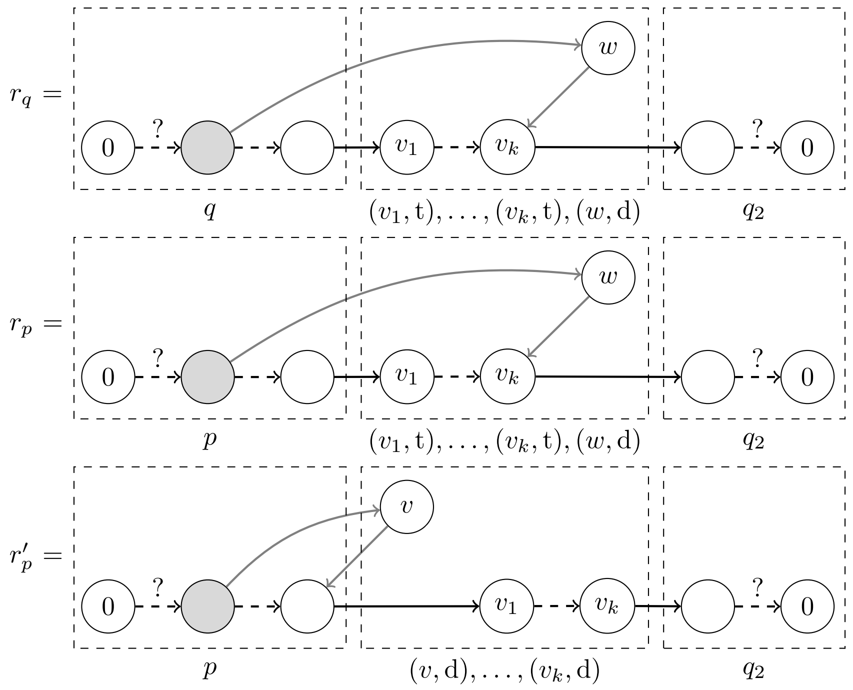
$r_q =$
?
$w$
$0$
$v_1$
$v_k$
$q$
$(v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d})$
$q_2$
$r_p =$
$p$
$r'_p =$
$v$
$(v, \mathrm{d}), \ldots, (v_k, \mathrm{d})$

Figure 3: Schemes for the dominance relation via $w$ of $p$ and $q$, possibly forking to $v$, and for the self-dominance via $v = w$ of $q + (v_1, \mathrm{t})$. Here, $r_p$ corresponds to Definition 3, whereas $r'_p$ corresponds to Definitions 4 and 5, and a gray vertex indicates the position of the drone in $p$ and $q$. Hence, $r_q$ is dominated (resp. self-dominated) via $w$ if either $r_p$ or $r'_p$ is $ng$-feasible and has a $u$-cost not greater than $r_q$.

$F^{\mathrm{D}}_{\mathrm{lbl}}(q) = V_c$, as the drone cannot leave $d(q)$. Together, the following rules generalize Dominance Rule 1; note that $F^{\mathrm{D}}$ is used instead of $F^{\mathrm{D}}_{\mathrm{lbl}}$ because these rules are correct regardless of the execution of the labeling algorithm.

**Dominance Rule 2.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is inside the truck. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, and $\bar{c}(p) \leq \bar{c}(q)$, then $p$ dominates $q$.*

**Dominance Rule 3.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $w \notin F^{\mathrm{D}}(p)$, $\bar{\tau}(p) \leq \bar{\tau}(q)$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), w) \leq \bar{c}(q) + c^{\mathrm{F}}(d(q), w)$, then $p$ dominates $q$ via $w$.*

**Dominance Rule 4.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $w \notin F^{\mathrm{D}}(p)$, $\bar{\tau}(p) \leq \bar{\tau}(q)$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, t(p)) \leq \bar{\tau}(q)$, then $p$ forking to $w$ dominates $q$ via $w$.*

Rules 3 and 4 have two advantages when compared against Rule 1. First, the dominance relation is relaxed to a single vertex $w$. Second, the drone is not required to wait at the same position in both $p$ and $q$. Still, the dominance via $w$ is due to a move of the drone to $w$. The following new rule allows the dominance via $w$ because of a fork to a vertex $v \neq w$. Despite technicalities, the conditions in Rule 5 enforce that $v$ can be immediately visited by the drone after $p$ without affecting $F^{\mathrm{T}}(p)$, and that the cost of the extension does not surpass the cost $\bar{\tau}(q)$ already paid by $q$.

**Dominance Rule 5.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, $v \in V_c$, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $v \notin F^{\mathrm{D}}(p) \cup N(t(p))$, $\bar{\tau}(p) - u_v \leq \bar{\tau}(q) - u_w$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), v) + c^{\mathrm{J}}(v, t(p)) - u_v \leq \bar{\tau}(q) - u_w$, then $p$ forking to $v$ dominates $q$ via $w$.*

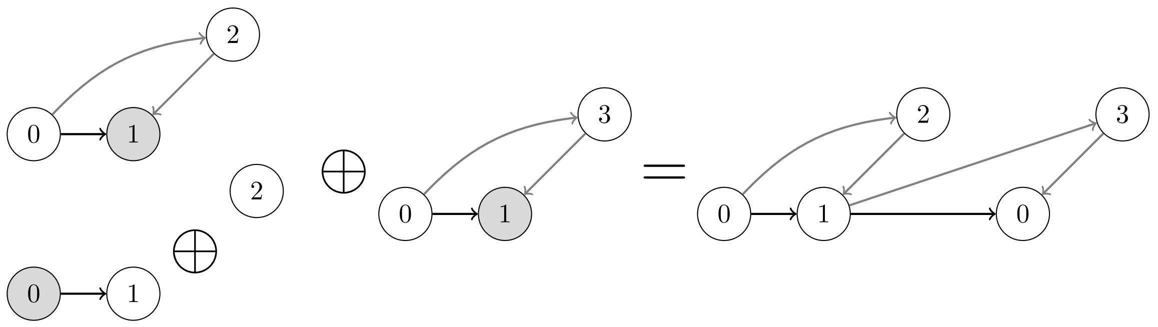

Figure 4: Two ways of merging partial routes to generate the same route $(0,\mathrm{d}),(1,\mathrm{t}),(2,\mathrm{d}),(0,\mathrm{t}),(3,\mathrm{d}) = (0,\mathrm{d}),(1,\mathrm{t}),(2,\mathrm{d}) \oplus (0,\mathrm{d}),(1,\mathrm{t}),(3,\mathrm{d}) = (0,\mathrm{d}),(1,\mathrm{t}) \oplus (2,\mathrm{d}) \oplus (0,\mathrm{d}),(1,\mathrm{t}),(3,\mathrm{d})$ in which the truck forwardly traverses the path $0,1,0$ and the drone forwardly traverses the path $0,2,1,3,0$. Observe that the moves defining the backward route are not simply reversed when the forward route is defined.

It is important to note that Rule 5 can be applied to $q = p$ to dominate the extension via $w$ because of $p$ forking to $v \neq w$. To further reduce the number of explored states, we also discard those partial routes that are self-dominated according to the following definition (Figure 3).

**Definition 5.** *Let $p$ be an ng-feasible route in which the drone is outside the truck, $v_1 \in V_c \setminus F^{\mathrm{T}}(p)$, and $w \in V_c$. We say that $p + (v_1, \mathrm{t})$ is* self-dominated *via $w$ when $r'_p = p + (w,\mathrm{d}),(v_1,\mathrm{d}),\ldots,(v_k,\mathrm{d}) + q_2$ is an ng-feasible route with $\bar{c}(r'_p) \leq \bar{c}(r_q)$, for every ng-feasible route $r_q = p + (v_1,\mathrm{t}),\ldots,(v_k,\mathrm{t}),(w,\mathrm{d}) + q_2$.*

As before, if we detect that $p$ is self-dominated via $w$, then we insert $w$ into $F^{\mathrm{D}}_{\mathrm{lbl}}(p)$ to discard each route of the form $p + (v_2,\mathrm{t}),\ldots,(v_k,\mathrm{t}),(w,\mathrm{d})$. The rule we use to detect self-dominance is the following.

**Dominance Rule 6.** *Let $p$ be an ng-feasible partial route in which the drone is outside the truck, $v \in V \setminus F^{\mathrm{T}}(p)$, and $w \in V_c \setminus F^{\mathrm{D}}(p)$. If $w \notin N(t(p))$, and $c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, t(p)) \leq \bar{\tau}(p + (v,\mathrm{t}))$, then $p + (v,\mathrm{t})$ is self-dominated via $w$.*

## 3.1 Bidirectional Search

The labeling algorithm solves the pricing problem using a *forward search* in which every partial route is extended in the direction of the arcs of the transport network $D$. An alternative method is to solve each pricing problem using a *bidirectional search* (Righini and Salani, 2006). Let $D^{-1}$ be the transport network that is obtained from $D$ by reversing its costs; that is, $c^{\mathrm{T}}_{D^{-1}}(i,j) = c^{\mathrm{T}}_{D}(j,i)$, $c^{\mathrm{F}}_{D^{-1}}(i,j) = c^{\mathrm{J}}_{D}(j,i)$, and $c^{\mathrm{J}}_{D^{-1}}(i,j) = c^{\mathrm{F}}_{D}(j,i)$ for every arc $ij$ of $D$. In a bidirectional search, the idea is to compute routes of $D$ by merging partial routes of $D$ and $D^{-1}$. There are two cases to consider for the merge of a partial route $p$ of $D$ and a partial route $q$ of $D^{-1}$ (Figure 4). Let $t^p_1, \ldots t^p_i$ and $d^p_1, \ldots d^p_j$ be the paths followed by the truck and the drone in $p$, respectively, and $t^q_1, \ldots t^q_k$ and $d^q_1, \ldots d^q_\ell$ be the paths followed by the truck and the drone in $q$, respectively. On the one hand, if $t^p_i = t^q_k$ and the drone is inside the truck in both $p$ and $q$, then the *merge* of $p$ and $q$ is the partial route $r = p \oplus q$ in which the truck and drone follow the paths $t^p_1, \ldots, t^p_i, t^q_{k-1}, \ldots, t^q_1$ and $d^p_1, \ldots, d^p_j, d^q_{\ell-1}, \ldots, d^q_1$, respectively. On the other hand, if $t^p_i = t^p_k$ and the drone is outside the truck in $p$, then the *merge* of $p$ and $q$ *via* $v \in V_c \setminus \{d^p_j, d^q_\ell\}$ is the partial route $r = p \oplus (v,\mathrm{d}) \oplus q$ in which the truck and drone follow the paths $t^p_1, \ldots, t^p_i, t^q_{k-1}, \ldots, t^q_1$ and $d^p_1, \ldots, d^p_j, v, d^q_\ell, \ldots, d^q_1$, respectively. We remark that the definition of the merge is not symmetric: the drone may be inside or outside the truck in $q$ in the second case. For the sake of uniformity, we may write $p \oplus (t(p),\mathrm{t}) \oplus q$ instead of $p \oplus q$ in the former case.

Observation 1 describes each route of $D$ as the result of a merge between a partial route $p$ of $D$ and a partial route $q$ of $D^{-1}$. Furthermore, it shows how the $u$-cost $\bar{c}(r)$ of $r$ can be obtained in $O(1)$ time when $p$ and $q$ are given.

**Observation 1.** *For every route $r$ of $D$ and every $0 \leq m \leq n+1$ there exists a unique partial route $p$ of $D$, a unique partial route $q$ of $D^{-1}$, a unique $v \in V$, and a unique $x \in \{\mathrm{d}, \mathrm{t}\}$ such that $|p| = m$ and $r = p \oplus (v, x) \oplus q$. Moreover, $\bar{c}_D(r) = \bar{c}_D(p) + \bar{c}_{D^{-1}}(q) - u_{t(p)} + \max\{\tau_D(p) + \tau_{D^{-1}}(q), c^{\mathrm{F}}_D(d(p), v) + c^{\mathrm{F}}_{D^{-1}}(d(q), v)\}$.*

Recall that the goal in the pricing problem is to find an *ng*-fesible route with a minimum $u$-cost. Applying the labeling algorithm on $D$ and $D^{-1}$, we can find two families of *ng*-feasible partial routes to be merged. Still, we need to decide whether the merge of two *ng*-feasible partial routes yields an *ng*-feasible route. Moreover, we need to make sure that at least one *ng*-feasible route with a minimum $u$-cost can be generated by merging two *ng*-feasible partial routes. To address these issues, we characterize the notion of *ng*-feasibility in terms of two partial routes that are merged together.

Fix $1 \leq m \leq n+2$ and let $r = p \oplus (v, x) \oplus q$ be a route of $D$ with $|p| = m$. We say that $r$ is *ng-feasible at* $m$ when $p$ and $q$ are *ng*-feasible, $F^{\mathrm{T}}(p) \cap F^{\mathrm{T}}(q) = \emptyset$ and either $x = \mathrm{t}$ or $v \notin F^{\mathrm{D}}(p) \cup F^{\mathrm{D}}(q)$. It is easy to see that every *ng*-feasible route $r$ is *ng*-feasible at $n+2$, because $r = r \oplus (0, \mathrm{d})$. The next proposition generalizes *ng*-feasibility to every possible value of $m$.

**Theorem 1.** *The following conditions are equivalent for a route $r$ of $D$.*

1. *$r$ is ng-feasible,*
2. *$r$ is ng-feasible at some $1 \leq m \leq n+2$, and*
3. *$r$ is ng-feasible at every $1 \leq m \leq n+2$.*

**Corollary 1.** *A route $r$ of $D$ is ng-feasible if and only if its corresponding route $(0, \mathrm{d}) \oplus r$ of $D^{-1}$ is ng-feasible.*

Given $1 < m < n+2$, the *bidirectional labeling algorithm* solves the pricing problem in three main steps. The *forward step* applies the labeling algorithm on $D$ to obtain a family $\mathcal{F}$ of *ng*-feasible partial routes with length $m$. The *backward step* applies the same algorithm on $D^{-1}$ to obtain a family $\mathcal{B}$ of *ng*-feasible partial routes with length either $n+3-m$ or $n+2-m$. Finally, the *merge step* computes the family $\mathcal{R}$ of routes $p \oplus (v, x) \oplus q$ for every $p \in \mathcal{F}$, every $q \in \mathcal{B}$ with $F^{\mathrm{T}}(p) \cap F^{\mathrm{T}}(q) = \emptyset$, every $x \in \{\mathrm{t}, \mathrm{d}\}$, and every $v \in V$ such that either:

($m_1$) $x = \mathrm{t}$ and $v = t(p)$,

($m_2$) $x = \mathrm{d}$, the drone is inside the truck in $q$, and $v \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p) \cup F^{\mathrm{D}}_{\mathrm{lbl}}(q)$, or

($m_3$) $x = \mathrm{d}$, the drone is outside the truck in $q$, and $v \notin F^{\mathrm{D}}(p) \cup F^{\mathrm{D}}_{\mathrm{lbl}}(q)$.

To accelerate the process, at most 200 routes are computed.

Theorem 2 below implies that the bidirectional labeling algorithm is correct. We remark that ($m_3$) not symmetric, as it considers $F^{\mathrm{D}}(p)$ and $F^{\mathrm{D}}_{\mathrm{lbl}}(q)$. Recall that $F^{\mathrm{D}}_{\mathrm{lbl}}(\cdot)$ is the set of customers that the drone cannot visit next because of *ng*-feasibility or dominance. The asymmetry of ($m_3$) arises because Rule 5 may forbid $w$ in the forward step because of a fork to $v \neq w$ while it can also forbid $v$ because of $w$ in the backward step. In such a case, both routes $p \oplus (v, \mathrm{d}) \oplus q$ and $p \oplus (w, \mathrm{d}) \oplus q$ are missed at the merge step if $v$ is required to be outside $F^{\mathrm{D}}_{\mathrm{lbl}}(p) \cup F^{\mathrm{D}}_{\mathrm{lbl}}(q)$. On the contrary, the algorithm is correct when every vertex outside $F^{\mathrm{D}}(p) \cup F^{\mathrm{D}}(q)$ is considered; we avoid this simpler algorithm because it is less efficient.

**Theorem 2.** *The family $\mathcal{R}$ computed by the bidirectional search contains at least one ng-feasible route with a minimum u-cost.*

## 3.2 Completion Bounds

Within a labeling algorithm executed on a transport network $D$, a completion bound of a partial route $p$ is a lower bound of the minimum $u$-cost that the labeling algorithm can achieve by extending $p$. In this work, the completion bound of $p$ is defined in terms of the family $\mathcal{B}$ of *ng*-feasible partial routes that the labeling

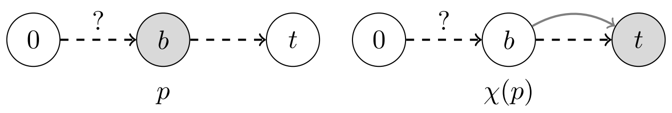

Figure 5: The partial route $\chi(p)$ associated to a partial route $p$. Here $b$ is the vertex at the bifurcation point and $t$ the last vertex visited by the truck.

algorithm computes when applied on $D^{-1}$. Observe that $p$ has an *associated* partial route $\chi(p)$ that has the same truck and drone paths as $p$, except that the truck carries the drone after the bifurcation point of $p$ (Figure 5); certainly, $p = \chi(p)$ when the drone is inside the truck in $p$. In terms of the sequence of moves, $\chi(p)$ is obtained from $p$ by replacing $(v, \mathrm{t})$ with $(v, \mathrm{d})$ for every move $(v, \mathrm{t})$ that appears after the bifurcation point $b(p)$. The *completion bound* $cb_{\mathcal{B}}(p)$ of $p$ is defined as follows:

1. if the drone is inside the truck in $p$, then $cb_{\mathcal{B}}(p) = \min\{\bar{c}(r) \mid r = p \oplus q \text{ is } ng\text{-feasible for } q \in \mathcal{B}\}$,
2. Otherwise, then $cb_{\mathcal{B}}(p) = \min\{cb_1, cb_2, cb_{\mathcal{B}}(\chi(p))\}$, where

$$cb_1 = \min\left\{\bar{c}(r) \;\middle|\; \begin{array}{l} r = p \oplus (v, \mathrm{d}) \oplus q \text{ is } ng\text{-feasible for a partial route } q \in \mathcal{B} \text{ such that} \\ \text{the drone is inside the truck, and } v \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p) \cup F^{\mathrm{D}}_{\mathrm{lbl}}(q) \end{array}\right\}, \text{ and}$$

$$cb_2 = \min\left\{\bar{c}(r) \;\middle|\; \begin{array}{l} r = p \oplus (v, \mathrm{d}) \oplus q \text{ is } ng\text{-feasible for a partial route } q \in \mathcal{B} \text{ such that} \\ \text{the drone is outside the truck, and } v \notin F^{\mathrm{D}}(p) \cup F^{\mathrm{D}}_{\mathrm{lbl}}(q) \end{array}\right\}.$$

As usual, we omit $\mathcal{B}$ when it is clear by context. As stated above, $cb(p)$ must be a lower bound of the minimum $u$-cost that the labeling algorithm can achieve by extending $p$. In many routing problems (e.g. Baldacci et al., 2012), $cb(p)$ is defined as the minimum $u$-cost that can be obtained when $p$ continues its journey through some partial route $q$ of $\mathcal{B}$. Unfortunately, this definition is flawed in our context when the drone is outside the truck in $p$: due to our dominance rules, $q$ need not exist or, even worse, the merge of $p$ and $q$ could have a greater $u$-cost than what the labeling algorithm can find. This is the reason why the completion bound of $p$ takes $\chi(p)$ into account. Also, note that $F^{\mathrm{D}}(p)$ is considered instead of $F^{\mathrm{D}}_{\mathrm{lbl}}(p)$ when the drone is outside the truck in $q$ because, as explained in Section 3.1, we cannot assure that the same customers are forbidden by the forward and backward algorithms. The fact that $cb(p)$ is a completion bound of $p$ follows from the next proposition.

**Proposition 1.** *Let $\mathcal{B}$ be the family of ng-feasible routes that is obtained by running the labeling algorithm on $D^{-1}$. If $r = p \oplus (v, x) \oplus q$ is an ng-feasible route of $D$, and $v \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p)$ when $x = \mathrm{t}$, then $cb_{\mathcal{B}}(p) \leq \bar{c}(r)$.*

## 3.3 Two Heuristics for the Pricing Problem

The bidirectional labeling algorithm can be easily adapted into two faster methods that heuristically search for a route with a negative $u$-cost. In a nutshell, the idea is to apply stronger dominance rules that discard non-promising partial routes. These methods are heuristic because every route with a minimum $u$-cost is discarded when some of its partial routes is not promising, i.e., Theorem 2 does not hold for these heuristics. *Cost-dominance* adapts Rule 2 to discard a partial route $p$ even if the drone is outside the truck in $p$. *Cost-ng-dominance* further adapts Rule 2 to relax the condition that $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$.

**Definition 6** (cost-dominance)**.** *Let $p$ and $q$ be ng-feasible partial routes. We say that $p$* cost-dominates *$q$ when $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, and $\bar{\tau}(p) \leq \bar{\tau}(q)$.*

**Definition 7** (cost-*ng*-dominance)**.** *Let $p$ and $q$ be ng-feasible partial routes. We say that $p$* cost-*ng*-dominates *$q$ when $|p| = |q|$, $t(p) = t(q)$, and $\bar{\tau}(p) \leq \bar{\tau}(q)$.*

The *cost-labeling* and *cost-ng-labeling* heuristic are adaptations of the bidirectional labeling algorithm in which $q$ is discarded if it is detected to be cost-dominated and cost-*ng*-dominated by $q$, respectively. For these heuristics, we limit the number of computed routes to 50.

## 4 Neighborhood Augmentation

Recall from Section 2 that the TSP-D on a transport network $D$ is solved via an integer program $\mathrm{MP}(D, \Omega)$, where $\Omega$ is a family of *ng*-feasible routes. As part of our solution, we implement a DSSR (Righini and Salani, 2008) that applies a *neighborhood augmentation* (NA) method similar to those of Tilk and Irnich (2017) and Lera-Romero et al. (2022). Given a sequence of non-elementary routes $\mathcal{R}$, the NA method computes a new family of *ng*-feasible routes $\Omega' \subset \Omega$ that does not contain the first route $r^*$ of $\mathcal{R}$. The routes in $\mathcal{R} \setminus \{r^*\}$ are used to further restrict $\Omega$, but eventually they could belong to $\Omega'$. Recall that $\Omega$ is implicitly defined through a neighborhood $N(v)$, for each $v \in V$ (Section 2.1). To compute $\Omega'$, the NA method defines a new neighborhood $N'(v) \supseteq N(v)$ for each $v \in V$ in such a way that $r^*$ is not *ng*-feasible with respect to $N'$.

The NA method considers one route $r \in \mathcal{R}$ at a time. Recall that $r$ is described by a sequence of moves $(v_1, x_1), \ldots, (v_{n+2}, x_{n+2})$. Since $r$ is not elementary, it contains a subsequence $(v_i, x_i), \ldots, (v_j, x_j)$, $j > i$, such that $v_i = v_j$. Let $p$ be the partial route $(v_1, x_1), \ldots, (v_{j-1}, x_{j-1})$, and recall that the bifurcation point $b(p)$ is the greatest index such that $x_{b(p)} = \mathrm{d}$. Let

$$e = \begin{cases} j-1 & \text{if the drone is inside the truck in } p \\ b(p) & \text{otherwise.} \end{cases}$$

We say that $C = v_i, \ldots, v_e$ is *cycle* of $r$ when all the vertices in $C$ are pairwise different. By Definition 1, $r$ is not *ng*-feasible with respect to $N'$ when $v_i \in N'(v_p)$ for every $i < p \leq e$. Thus, to exclude $r$ from $\Omega$, as well as all the other routes that have $C$ as a cycle, it suffices to insert $v_i$ into $N(v_p)$ for every $i < p \leq e$. Such a process is referred to as the *separation* of $C$.

Although we can separate all the cycles of every route in $\mathcal{R}$, doing so increases the size of the neighborhoods too fast, making each subsequent pricing problem intractable. Therefore, the NA method limits the effects of the separation as follows. Let $s = \max\{|N(v)| \mid v \in V\}$. For each $r \in \mathcal{R}$, every cycle $C = v_1, \ldots, v_j$ of $r$ is processed in a non-decreasing order of $|C|$. If $|N(v_i \cup \{v_1\})| \leq s + 1$ for every $1 < i \leq j$ when $C$ is processed, then $C$ is separated. Obviously, the NA method separates at least one cycle of $r^*$.

## 5 Variable Fixing

As stated in Section 2, to solve the TSP-D we compute a sequence of lower bounds $lb_0 < \ldots < lb_k = c^*$ of the optimum value $c^*$. Given a "candidate bound" $ub \in \mathbb{R}$, a major step of our solver is to decide if $ub$ is a strict lower bound of $c^*$. We implement a variable fixing method to reduce the time required for this step. In a nutshell, the idea is to fix to $\infty$ the cost of those arcs of the transport network $D$ that are not traversed by an elementary route $r$ with $c(r) < ub$. Then, each time a labeling algorithm is executed, those partial routes that traverse some of these arcs are immediately discarded. In other words, the set of *ng*-feasible routes $\Omega$ with a cost $c < \infty$ is reduced to a smaller subset of routes that do not traverse the fixed arcs.

### 5.1 Forward fixing

Recall that each instance of the TSP-D has a positive weight $c^{\mathrm{T}}_D(i, j)$ that denotes the time required to travel from vertex $i$ to vertex $j$. Certainly, we can generalize the TSP-D considering two positive weights $c^{\mathrm{T}}_{D,\mathrm{t}}(i, j, k)$ and $c^{\mathrm{T}}_{D,\mathrm{d}}(i, j, k)$ that denote the time required to travel from $i$ to $j$ at the $k$-th move: the former is taken when the truck travels alone, and the latter when the truck carries the drone. We can further generalize the TSP-D using the functions $c^{\mathrm{F}}_D(i, j, k)$ and $c^{\mathrm{J}}_D(i, j, k)$ instead of $c^{\mathrm{F}}_D(i, j)$ and $c^{\mathrm{J}}_D(i, j)$, respectively. The former denotes the time required by the drone to leave the truck at $i$ to visit $j$ from the bifurcation point $k$, while the latter denotes the time spent by the drone to meet the truck again at $j$ after visiting $i$ alone at the $k$-th move. Figure 6 depicts the new costs for a generic route; as usual, $D$ is omitted when it is clear by context. We leave it to the reader to observe that the algorithms developed thus far are correct and equally efficient for the generalized TSP-D. We remark that, to do so, the costs for the reversed network $D^{-1}$ must be defined. For instance, if at the $k$-th move of a route $r$ of $D$ the drone visits $v$ alone and then meets

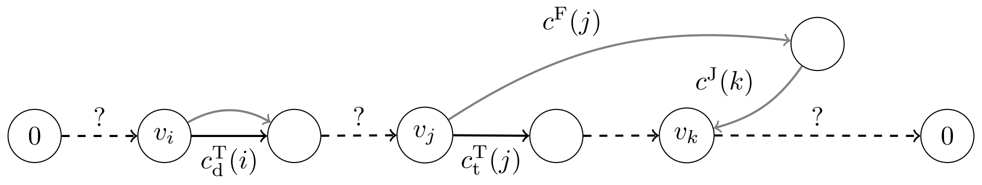
$c^{\mathrm{F}}(j)$
$c^{\mathrm{J}}(k)$
0
?
$v_i$
$c^{\mathrm{T}}_{\mathrm{d}}(i)$
?
$v_j$
$c^{\mathrm{T}}_{\mathrm{t}}(j)$
$v_k$
?
0

Figure 6: Generalization of the TSP-D with costs that depend on moment in which an arc is traversed, taking into account that $v_h$ is traversed after move $h-1$ and before move $h$, for $h \in \{i,j,k\}$. For clarity, we omit the vertex parameters from the cost functions are they are clear by the figure

the truck at $w$, then the drone leaves $w$ at the bifurcation point $n+1-k$ to visit $v$ in the route of $D^{-1}$ corresponding to $r$. Hence, $c^{\mathrm{J}}_{D^{-1}}(i,j,k) = c^{\mathrm{F}}_{D}(j,i,n+1-k)$ whereas $c^{\mathrm{F}}_{D^{-1}}(i,j,k) = c^{\mathrm{J}}_{D}(j,i,n+1-k)$.

The fixing method updates some values of $c^{\mathrm{T}}_{\mathrm{t}}$, $c^{\mathrm{T}}_{\mathrm{d}}$, $c^{\mathrm{F}}$, and $c^{\mathrm{J}}$ to $\infty$ to forbid the traversal of some arcs at certain moves or bifurcation points. To decide which arcs to update, the fixing method solves a pricing problem using a two-step algorithm called the labeling algorithm *with upper bound ub*, for a given $ub \in \mathbb{R}$. In a first step, the labeling algorithm is applied to $D^{-1}$ to obtain a family $\mathcal{B}$ of partial routes. In a second step, a variant of the labeling algorithm with Rules 4–6 disabled, that immediately discards every partial route with completion bound $cb \geq ub$, is executed on $D$ to obtain a family of partial routes $\mathcal{F}$. Of course, the second step is initialized with $\mathcal{B}$ and it computes $cb(p)$ for every partial route.

**Proposition 2.** *For $1 \leq m \leq n+2$, let $\mathcal{F}_m$ be the family of partial routes of length $m$ obtained when the labeling algorithm with upper bound ub is executed on $D$. If $r = (p + (v, x_v)) \oplus (w, x_w) \oplus q$ is an elementary route with $\bar{c}(r) < ub$, then there exists $p' \in \mathcal{F}_{|p|}$ with $t(p') = t(p)$ such that $r' = (p' + (v, x_v)) \oplus (w, x_w) \oplus q$ is ng-feasible and $cb(p' + (v, x_v)) < ub$.*

The *forward fixing* method has two steps. In the first step, the labeling algorithm with upper bound $ub$ is run on $D$ to obtain the family $\mathcal{F}_m$ of partial routes with length $m$ for every $1 \leq m \leq n+2$. The second step traverses each arc $vw$ of $D$ and every $1 \leq m \leq n+1$. By Proposition 2, if no *ng*-feasible partial route $p + (w, \mathrm{d})$ exists with $p \in \mathcal{F}_m$, $t(p) = v$, and $cb(p + (w, \mathrm{d})) < ub$ such that the drone is inside the truck at $p$, then every elementary route $r$ in which the truck carries the drone from $v$ to $w$ at the $m$-th move has $\bar{c}(r) \geq ub$. Therefore, the truck-only variable fixing method sets $c^{\mathrm{T}}_{D,\mathrm{d}}(v,w,m) = \infty$ to exclude all such elementary routes. Similarly, the forward fixing method sets: $c^{\mathrm{T}}_{D,\mathrm{t}}(v,w,m) = \infty$ if no *ng*-feasible partial route $p + (w, t)$ exists with $p \in \mathcal{F}_m$, $t(p) = v$, and $cb(p + (w, \mathrm{t})) < ub$; and $c^{\mathrm{J}}(v,w,m) = \infty$ if no *ng*-feasible partial route $p + (v, \mathrm{d})$ exists with $p \in \mathcal{F}_m$, $t(p) = w$, and $cb(p + (v, \mathrm{d})) < ub$.

The efficiency of the fixing method depends on how large is $ub - \overline{lb}$, as more labels are discarded when the $u$-bound $\overline{lb}$ is closer to $ub$. Thus, even though Rules 4–6 are disabled, the second step of the labeling algorithm runs faster than the first one when $\overline{lb} \approx ub$. On the contrary, if $\overline{lb} \ll ub$, then fewer labels get discarded and, consequently, the fixing method can consume a lot of time. Summing up, the fixing method should be avoided when the $u$-bound $\overline{lb}$ is far from $ub$ because it is too time consuming. Even worse, the method is ineffective in this case, as fewer arcs are fixed when fewer partial routes get discarded by completion bounds.

## 5.2 Two-pass fixing

The forward fixing method is asymmetric, in the sense that it does not update $c^{\mathrm{F}}$. However, every update of $c^{\mathrm{J}}$ in $D$ corresponds to an update of $c^{\mathrm{F}}$ in $D^{-1}$. Consequently, to remove as many arcs as possible, one possibility is to run the method twice, first on $D^{-1}$ an then on $D$. We can obtain a faster and/or more effective algorithm for the second pass if we take advantage of the data computed in the first pass.

To improve the speed of the second pass, recall that the first step of the second pass runs the labeling algorithm on $D^{-1}$ to obtain a family of partial routes $\mathcal{B}$ that is later used to compute completion bounds. The execution of the labeling algorithm on $D^{-1}$ can be avoided, observing that the first pass runs a (forward) labeling algorithm that solves the pricing problem defined by $D^{-1}$, $\Omega$, and $u$. That is, the first pass produces a family of partial routes $\mathcal{B}'$ that replaces $\mathcal{B}$ when completion bounds are computed in the second pass. By definition, $\mathcal{B}'$ is the subset of $\mathcal{B}$ that is obtained by discarding every partial route $p \in \mathcal{B}$ with $cb(p) \geq ub$. Hence, the second pass is still correct, because $p \in \mathcal{B} \setminus \mathcal{B}'$ cannot be used to produce a completion bound less than $ub$ when the forward fixing is run on $D$. The new approach saves time not only because the execution of the labeling algorithm on $D^{-1}$ is avoided, but also because fewer partial routes of $D^{-1}$ are considered each time the completion bound of a partial route of $D$ is computed in the second pass.

To improve the effectiveness of the second pass, note that the first pass produces a family of *ng*-feasible routes $\mathcal{R}$ that contains at least one route $r^*$ with the minimum reduced cost. Hence, immediately after the first pass, we can run the neighborhood augmentation algorithm (Section 4) to restrict the family of *ng*-feasible routes considered to $\Omega' \subseteq \Omega \setminus \{r^*\}$. The second pass is correct because every partial route in $\Omega'$ is *ng*-feasible for the neighborhoods employed in the first pass. Consequently, Propositions 1 and 2 hold for the second pass. Furthermore, the second pass is a labeling algorithm that solves a pricing problem on the more restricted set of routes $\Omega'$, thus we obtain a better $u$-bound.

The *two-pass fixing* method is obtained by combining both of the approaches above. First, a forward fixing is executed on $D^{-1}$ as described in Section 5.1. A family of partial routes $\mathcal{B}$ and a family of *ng*-feasible routes $\mathcal{R}$ are obtained in this first step. Then, the neighborhood augmentation algorithm is invoked with input $\mathcal{R}$ to restrict $\Omega$ into $\Omega'$. Finally, the forward fixing method is executed for $D$ and $\Omega'$, using $\mathcal{B}$ for the computation of the completion bounds. We remark that $\Omega'$ is used only for the purposes of variable fixing. Once the fixing method is concluded, the invoking algorithm goes on using $\Omega$ as the set of *ng*-feasible routes.

## 5.3 Iterative fixing

Instead of running two passes of the forward fixing method, we may run several passes to obtain the *iterative fixing* method. The first iteration runs the forward fixing method on $D^{-1}$. Then, each iteration $i > 1$ runs a forward fixing on $D$ if $i$ is even or on $D^{-1}$ if $i$ is odd. After each iteration, a family of routes $\mathcal{R}_i$ and a family of partial routes $\mathcal{B}_i$ are obtained. As in the two-pass fixing method, $\mathcal{R}_i$ is used to restrict the notion of *ng*-feasibility with the neighborhood augmentation algorithm of Section 4, whereas $\mathcal{B}_i$ is used to compute the completion bounds within the $(i+1)$-th execution of the forward fixing method. The algorithm applies at least two iterations and it stops when at least one of the two last iterations required more than max-fix-time time, for a parameter max-fix-time. The correctness of the iterative fixing method follows by induction on $i$ with the same arguments used for the two-pass fixing method.

Iterative fixing gives us a tool to decide if $ub$ is a lower or upper bound of the optimum value $c^*$ of the TSP-D. Indeed, if max-fix-time $= \infty$ and $\Omega_i$ is the set of routes used for iteration $i$, then there exists some $k$ such that $\Omega_k$ contains only elementary routes because $\Omega_i$ is a proper subset of $\Omega_{i-1}$ for every $i > 1$. If $ub > c^*$, then an optimum route $r^*$ is found at some iteration $i < k$. Otherwise, a lower bound of $c^*$ not less than $ub$ is obtained at some iteration $i \leq k$. We remark that if $ub > c^*$, then the iterative fixing method solves the TSP-D, as $r^*$ is an elementary route with cost $c(r^*) = c^*$. In few words, iterative fixing applies a DSSR as defined by Righini and Salani (2008).

Iterative fixing can be a quite efficient method to decide if $ub$ is a lower or upper bound of $c^*$, specially when the $u$-bound $\overline{lb}(\Omega_1)$ is close to $ub$. This is because the completion bound of a partial route in the $(i+1)$-th iteration cannot be less than its completion bound in the $i$-th iteration. If the former tends to be higher than the latter, then more partial routes are discarded because their completion bounds surpass $ub$. Consequently, when the $(i+1)$-th iteration is compared against the $i$-th iteration, the number of new non-dominated routes that are enumerated tends to be much smaller than the number of new partial routes that can be discarded because of completion bounds. Altogether, this results in fewer partial routes being enumerated at each iteration. On the contrary, if $ub \gg \overline{lb}(\Omega_i)$ or the completion bounds of the partial routes do not increase fast enough, then the number of partial routes enumerated at the $(i+1)$-th iteration tends to be much higher than that of the $i$-th iteration.

### 5.4 First step of fixing

Both the two-pass and the iterative fixing methods require the execution of the labeling algorithm on the reverse transport network $D^{-1}$ as its very first step. We always execute a fixing method after the bidirectional labeling algorithm of Section 3.1 was applied on $D$. Thus, we readily have access to a family of partial routes $\mathcal{F}$ of $D$ and a family of partial routes $\mathcal{B}$ of $D^{-1}$. We take advantage of this fact to avoid the execution of the labeling algorithm on $D^{-1}$ from scratch. Instead, we continue the enumeration of the partial routes of $D^{-1}$ by extending the partial routes in $\mathcal{B}$. Moreover, we use $\mathcal{F}$ to compute the completion bound $cb_{\mathcal{F}}(p)$ of each enumerated partial route $p$ and we immediately discard $p$ if $cb_{\mathcal{F}}(p) \geq ub$. The family of backward routes thus obtained is used to compute the completion bounds in the first execution of the forward fixing method. As explained above, the algorithm is correct because a discarded partial routes of $D^{-1}$ cannot be used to produce a completion bound below $ub$ when it is merged with a partial route of $D$. As also explained before, this modified algorithm can be much faster than the execution of the labeling algorithm from scratch, specially when $ub$ is sufficiently tight.

# 6 Column Generation

As stated in Section 2, column generation is a main ingredient of our algorithm for the TSP-D. Hereafter, let $\text{LMP}(D, \Omega)$ be the linear relaxation of $\text{MP}(D, \Omega)$ for any transport network $D$ and any family of *ng*-feasible routes $\Omega$. Given $D$, $\Omega$ (implicitly defined via neighborhoods), and an *initial* family of routes $\Psi_0 \subseteq \Omega$, a column generation algorithm computes a family of routes $\Psi$ such that $\Psi_0 \subseteq \Psi \subseteq \Omega$ and the optimum $\ell$ of $\text{LMP}(D, \Psi)$ equals the optimum $\ell^*$ of $\text{LMP}(D, \Omega)$. In simpler words, and from a primal perspective, a column generation algorithm solves $\text{LMP}(D, \Omega)$, providing $\Psi$ as a certificate. Alternatively, and from a dual perspective, a column generation method computes a dual vector $u$ maximizing $\overline{lb}(D, u) = \ell = \ell^*$.

In this section we describe three column generation algorithms. The first one, called `CG`, is a generic algorithm that computes $\Psi$ and $u$ by repeatedly solving pricing problems. The second one, called `CG+F`, is an extension of `CG` that applies the two-pass fixing method from time to time, using a given *fixing bound*. The third one, called `B&CG+F`, works by repeatedly invoking `CG+F` while it "guesses" different fixing bounds. `B&CG+F` is usually faster than `CG` for the TSP-D, specially when the pricing problems are hard-to-solve, as it considers easier pricing problems to populate $\Psi$.

For the rest of this section, we assume that the initial family of routes $\Psi_0$ contains an elementary route with a fictitious huge cost, thus $\text{LMP}(D, \Psi)$ is always well defined for every $\Psi \supseteq \Psi_0$.

### 6.1 Generic Column Generation

Together with $D$, $\Omega$, and $\Psi_0$, `CG` takes a *reach factor* $\rho \in (0, 1]$. The purpose of `CG` is to compute a family of *ng*-feasible routes $\Psi \supseteq \Psi_0$ and a dual vector $u$ such that the $u$-bound $\overline{lb}(D, u)$ is not less than $\rho\ell$, where $\ell$ is the optimum of $\text{LMP}(D, \Psi)$. Observe that $\ell \geq \ell^*$ for the optimum $\ell^*$ of $\text{LMP}(D, \Omega)$, thus $\overline{lb}(D, u) \geq \rho\ell^*$. Moreover, if $\rho = 1$, then $\overline{lb}(D, u) = \ell = \ell^*$.

As any traditional column generation algorithm, `CG` is an iterative method that keeps a family of *ng*-feasible routes $\Psi \subseteq \Omega$ that defines the *restricted master problem* $\text{MP}(D, \Psi)$; initially $\Psi = \Psi_0$. Each iteration of `CG` starts by solving $\text{LMP}(D, \Psi)$, from which an upper bound $\ell$ of $\ell^*$ and a dual vector $u$ are obtained. Then, the pricing problem is solved to compute a set of routes $\mathcal{R}$ with a negative $u$-cost (i.e., a negative reduced cost). Finally, all the routes in $\mathcal{R}$ are inserted into $\Psi$. `CG` may also keep track of an upper bound of $c^*$; the elementary route of $\mathcal{R}$ with minimum cost, if any, can be used to update this upper bound.

We use a hierarchical approach to solve the pricing problems. Initially, the cost-*ng*-labeling heuristic of Section 3.3 is executed until it fails to find a route with a negative $u$-cost. Then, the cost-*ng*-labeling heuristic of Section 3.3 is applied, stopping when it fails to find a route with a negative $u$-cost. Finally, after both heuristics failed, the bidirectional labeling algorithm of Section 3.1 is executed. `CG` stops when either $\overline{lb}(D, u) \geq \rho\ell$ or the exact labeling algorithm fails to find a route with a negative $u$-cost. In the latter case, $\ell^* = \ell = \overline{lb}(D, u)$.

## 6.2 Column Generation with Variable Fixing

To reduce the time required to solve each pricing problem, CG+F extends CG to run the two-pass fixing method from time to time. Together with $D$, $\Omega$, and $\Psi_0$, CG+F receives a *fixing bound* $ub \geq 0$ and a number fix-gap $\in \mathbb{R}$; informally, fix-gap indicates when was the last time that the two-pass fixing method was applied on $D$. Observe that there is no way to indicate a reach factor $\rho$ as in CG, because CG+F extends CG only for the case $\rho = 1$. Three are the remaining differences between CG+F and CG. First, CG+F only applies the bidirectional labeling algorithm of Section 3.1 to solve each pricing problem. Second, each time a pricing problem is solved, CG+F checks if the current dual vector $u$ satisfies $ub - \overline{lb}(D, u) \leq 0.9\,\text{fix-gap}$. If so, then the two-pass fixing method is executed with upper bound $ub$ and fix-gap is updated to $ub - \overline{lb}$. Third, CG+F halts immediately if $\overline{lb}(D, u) \geq ub$.

Certainly, CG+F is a destructive algorithm that updates $D$ and fix-gap throughout its execution. The output of CG+F is a family of $ng$-feasible routes $\Psi \supseteq \Psi_0$ and a dual vector $u$ such that the $u$-bound $\overline{lb}(D, u)$ is either maximum or not less than $ub$. Note that $u$ fails to maximize $\overline{lb}(D_0, u)$, for the input network $D_0$, when every optimal solution to the pricing problem on $D_0$ takes an arc whose cost in the output network $D$ is fixed to $\infty$. To evaluate if $u$ maximizes $\overline{lb}(D_0, u)$, it is enough to solve the pricing problem defined by $u$ on the instance $D_0$ to test if $\bar{c}(D_0, u, r) \geq 0$ for the route $r$ with minimum $u$-cost. This is a sufficient condition, as $u$ can maximize $\overline{lb}(D_0, u)$ even though $D_0$ has routes with a negative $u$-cost.

We remark that the CG+F can be invoked with any fixing bound $ub \geq 0$. By Proposition 2, if $ub$ is an upper bound of the optimum $c^*$ of $\text{MP}(D_0, \Omega)$, then every optimum solution of the TSP-D in $D_0$ is an optimum solution of the TSP-D in $D$. On the contrary, if $ub < c^*$, then the TSP-D might became infeasible on $D$; moreover, the optimum solution on $D$, if any, need not be optimum on $D_0$. Nevertheless, CG+F is a well defined algorithm regardless of the value of $ub$. As described above, there are two possibilities for the output of CG+F:

**Possibility 1:** $\overline{lb}(D, u) < ub$ and $u$ maximizes $\overline{lb}(D, u)$, thus $\overline{lb}(D, u)$ is a lower bound of $c^*$.

**Possibility 2:** $\overline{lb}(D, u) \geq ub$, thus $ub$ is a lower bound of $c^*$.

The lower bound obtained in the former case may be tighter than those obtained when CG+F is executed with $ub \geq c^*$, as fewer costs of $D$ get fixed to $\infty$ when $ub \geq c^*$. Moreover, by the same reason, the lower bounds computed when $ub < c^*$ are obtained faster than when $ub \geq c^*$.

## 6.3 Bounded Column Generation with Variable Fixing

Given $D$, $\Omega$, and $\Psi_0$, the goal of B&CG+F is find a set of routes $\Psi \supseteq \Psi_0$ and a dual vector $u$ such that $\overline{lb}(D, u) = \ell^*$, where $\ell^*$ is the optimum of both $\text{LMP}(D, \Psi)$ and $\text{LMP}(D, \Omega)$. To do it so, B&CG+F computes a sequence of family of routes $\Psi_0 \subseteq \Psi_1 \subseteq \ldots \subseteq \Psi_k = \Psi$ and a sequence of dual vectors $u_1, \ldots, u_k = u$ such that $\overline{lb}(D, u_1) \leq \ldots \leq \overline{lb}(D, u_k) = \ell^*$.

As a first step, B&CG+F invokes CG with input $D$, $\Omega$, $\Psi_0$, and $\rho = 90\%$ to obtain the family of routes $\Psi_1$ and the dual vector $u_1$. As a by-product, B&CG+F also obtains the $u_1$-bound $ub_1 = \overline{lb}(D, u_1) \geq .9\ell^*$, while it records the time $\delta_1$ that CG consumed to solve its last pricing problem. The time $\delta_1$ is taken by B&CG+F as a measure of how hard is to solve a pricing problem on the input network $D$. Then, for $1 \leq i < k$, B&CG+F creates a private copy $D_{i+1}$ of $D$ and it applies three sub-steps to compute $\Psi_{i+1}$, $u_{i+1}$, and $\delta_{i+1}$. The first sub-step invokes the two-pass fixing method on $D_{i+1}$ using $ub_{i+1} = 1.005ub_i$ as the upper bound, to fix the costs of some arcs of $D_{i+1}$ to $\infty$. The second sub-step runs CG+F with input $D_{i+1}$, $\Omega$, $\Psi_i$, $ub = ub_{i+1}$, and fix-gap $= ub_{i+1} - \overline{lb}(D, u_i)$ to obtain the family of routes $\Psi_{i+1}$ and a dual vector $u$. This sub-step also fixes the costs of some arcs of $D_{i+1}$ to $\infty$. Finally, the third sub-step defines the dual vector $u_{i+1}$ and the time measure $\delta_{i+1}$, as follows. Let $\delta$ be the total time required by the first two sub-steps. If $\overline{lb}(D_{i+1}, u) \geq ub_{i+1}$ and $\delta < \delta_i$, then B&CG+F discards $u$ and it sets $u_{i+1} = u_i$ and $\delta_{i+1} = \delta_i$. Otherwise, B&CG+F solves the pricing problem defined by $u$ on the input network $D$ to obtain $\overline{lb}(D, u)$. Having solved a new pricing problem on $D$ in $\delta'$ time, B&CG+F sets $\delta_{i+1} = \max\{\delta_i, \delta'\}$ as a new measure of how hard can a pricing problem be on $D$. Regarding $u_{i+1}$, if $\overline{lb}(D, u) < \overline{lb}(D, u_i)$, then B&CG+F discards $u$ and it takes $u_{i+1} = u_i$; otherwise, B&CG+F

updates the dual vector setting $u_{i+1} = u$. The algorithm stops after the $k$-th iteration, with output $\Psi_k$, $u_k$, when it can guarantee that $u_k$ maximizes $\overline{lb}(D, u_k)$; for this, B&CG+F checks that $\bar{c}(D, u, r_k) \geq 0$ for the route $r$ of minimum $u_k$-cost.

It is easy to see that B&CG+F always ends because its unique stopping condition is eventually satisfied by a large enough $ub_k$. Moreover, $\overline{lb}(D, u) = \ell^*$ is the optimum value of LMP$(D, \Psi)$ for $\Psi_k$ and $u_k$. Hence, B&CG+F is correct. The efficiency of B&CG+F depends on the number of iterations $k$ and the time required to obtain $\Psi_i$ and $u_i$ at each iteration $1 \leq i \leq k$. Regarding $k$, observe that no more than 20 iterations are required to reach a fixing bound $ub_i \geq \ell^*$ because $ub_1 \geq 0.9\ell^*$. We expect $k$ to be not much greater than $i$ because, as $ub_i$ increases, there are fewer chances that every optimum solution of LMP$(D, \Omega)$ has a route passing through a fixed arc of $D_i$. Regarding the time $\delta$ required by the $i$-th iteration, we expect $\delta$ to be higher as $ub_i - \overline{lb}(D, u_i)$ increases. B&CG+F considers that $\delta$ is high enough when $\delta > \delta_i$, which happens when $ub_i$ is sufficiently far from $\overline{lb}(D, u_i)$. In such cases, B&CG+F computes $\overline{lb}(D, u)$, hoping it can update $u_{i+1}$ with a dual vector that provides a greater lower bound for the next iteration. For this step, B&CG+F expects to spend approximately $\delta_i$ time, which is less than the time required by the $i$-th iteration. Of course, this is a waste of time when $u$ does not improve $u_i$. However, the chances of updating $u_{i+1}$ get higher as the gap between $ub$ and $\overline{lb}(D, u_i)$ increases. Altogether, we expect B&CG+F to be much faster than CG because it solves easier pricing problems to find the routes in $\Psi_k$. This expectation is confirmed in practice, at least for the benchmark instances used in Section 8.

To end this section, we remark that B&CG+F also provides a lower bound $lb$ of the optimum $c^*$ of MP$(D, \Omega)$. This lower bound is the maximum of the lower bounds obtained at each execution of CG+F. As explained in Section 6.2, $lb$ is usually greater than $\ell^* = \overline{lb}(D, u)$, for the dual vector $u$ finally obtained by B&CG+F.

## 7 An Algorithm for the TSP-D

To solve the TSP-D, we implement solver called *price-fix-and-augment* (PFA) that follows a DSSR. From a high level perspective, PFA finds a sequence of lower bounds $lb_1 < \ldots < lb_k = c^*$ of the optimum value $c^*$. Together with $lb_i$, $1 \leq i \leq k$, PFA keeps a family of $ng$-feasible routes $\Omega_i$ (implicitly encoded by neighborhoods), a family of routes $\Psi_i \subseteq \Omega_i$, and a dual vector $u_i$. PFA also computes an optimal solution to the TSP-D when $i = k$. From a lower level perspective, the $i$-th iteration of PFA decides if a candidate bound $ub > lb_i$ is a lower or upper bound of $c^*$. Iterative fixing (Section 5.3) is the main tool to determine the status of $ub$, which is run on $\Omega_i$ and $u_i$ using $ub$ as the upper bound. The iterative fixing can be highly efficient, but only if $\overline{lb}(D, \Omega_i, u_i) \approx ub$. As $ub$ increases at each iteration, the value of $\overline{lb}(D, \Omega_i, u_i)$ must increase as well, for which PFA runs the B&CG+F algorithm (Section 6.3). Of course, the maximum value of $\overline{lb}(D, \Omega_i, u_i)$ depends on the size of $\Omega_i$, that is shrunk from time to time via the neighborhood augmentation method (Section 4). At each iteration $i$, PFA also records a measure $\delta_i$ of the time required by a pricing problem on $D$ and $\Omega_i$, that is used to decide about the efficiency of the iterative fixing step.

The input of PFA is a transport network $D$ and an integer value init-ng $\geq 0$. An initial family of $ng$-feasible routes $\Omega_0$ is obtained by defining a neighborhood $N(v)$, $v \in V_c$, that contains the init-ng customers closer to $v$. To find $lb_1$, PFA invokes B&CG+F with input $D$, $\Omega_0$, and $\Psi_0$, where $\Psi_0$ is a set whose unique element is the elementary route on which the truck visits the customers $1, \ldots, n$ in this order. As a by-product, the family of routes $\Psi_1$ and the dual vector $u_1$ are obtained, together with the $u_1$-bound $\overline{lb}(D, u_1)$ and the time $\delta_1$ consumed by B&CG+F to solve its last pricing problem. PFA concludes the first iteration setting $\Omega_1 = \Omega_0$.

For $i \geq 1$, PFA runs a *fix-augment-and-price* loop to determine the lower bound $lb_{i+1} = \min\{c^*, ub\}$, where $ub = 1.0025 lb_i$. Starting with $\Omega_{i+1} = \Omega_i$, $\Psi_{i+1} = \Psi_i$, and $\delta_{i+1} = \delta_i$, the loop applies the following steps, while it updates a private network $D_{i+1}$ and a private dual vector $u$; initially, $D_{i+1} = D$ and $u = u_i$.

**Fix Step.** In this step, PFA runs the iterative fixing on $D_{i+1}$ and $\Omega_{i+1}$ with the dual vector $u$, using $ub$ as the upper bound and max-fix-time $= \delta_{i+1}$ to limit the running time. The private dual vector $u$ is used for this step, as it gets updated by the other steps of the loop. We remark that the cost of some arcs of $D_{i+1}$ are fixed to $\infty$ as the iterative fixing method unfolds.

By invariant, $D_{i+1}$ is obtained from $D$ via variable fixing with the upper bound $ub$, whereas $\Omega_{i+1}$

contains all the elementary routes of $\Omega$. Consequently, $\mathrm{MP}(D_{i+1}, \Omega_{i+1})$ is infeasible or it has an optimum value $c' \geq c^*$. Moreover, in the latter case, either $c' = c^*$ or $ub < c^*$ (Proposition 2). Therefore, there are three possibilities for the iterative fixing method to stop:

**Possibility 1:** an elementary route $r^*$ solving $\mathrm{MP}(D_{i+1}, \Omega_{i+1})$ with cost $c(r^*) < ub$ is obtained. In this case $c(r^*) = c^*$, thus `PFA` stops with output $r^*$.

**Possibility 2:** $\mathrm{MP}(D_{i+1}, \Omega_{i+1})$ is infeasible or $ub < c'$. In this case, $lb_{i+1} = ub$ is a lower bound of $c^*$, `PFA` stops the fix-augment-and-price loop and the dual vector $u$ is obtained.

**Possibility 3:** the last fixing iteration required more than $\delta_{i+1}$ time. In this case, `PFA` considers $ub - \overline{lb}(D, \Omega_{i+1}, u)$ to be too high, thus it runs the Augment and Price Steps below to shrink $\Omega_{i+1}$ and to increase $\overline{lb}(D, \Omega_{i+1}, u)$, respectively.

**Augment Step.** In this step, `PFA` invokes the neighborhood augmentation (Section 4) algorithm to restrict the family of *ng*-feasible routes $\Omega_{i+1}$, using the sequence of routes $\mathcal{R}$ that was obtained when the pricing problem defined by $u$ was solved on $\Omega_{i+1}$ by `B&CG+F`. Also, all the routes of $\Psi_{i+1}$ that are not *ng*-feasible under the new neighborhoods are removed. Here, again, $u$ is the private dual vector. The reason for choosing $u$ instead of $u_{i+1}$, even if the cycles that get separated are not that meaningful for the input network, is that we prefer to increase $\overline{lb}(D_{i+1}, \Omega_{i+1}, u)$ as much as possible, as this gets us closer to decide the status of $ub$ as a bound of $c^*$.

**Price Step.** In this step, `PFA` runs `B&CG+F` with input $D_{i+1}$, $\Omega_{i+1}$, and $\Psi_{i+1}$ to update both $\Psi_{i+1}$ and the dual vector $u$. However, instead of running `B&CG+F` to completion, `PFA` stops `B&CG+F` the moment it reaches a fixing bound greater than $ub$. The reason why `B&CG+F` is invoked on $D_{i+1}$ instead of $D$ is because the Fix Step already fixed the cost of some arcs of $D_{i+1}$, thus `B&CG+F` is faster on $D_{i+1}$ than on $D$. Also, our first task is to determine if $ub$ is a lower or upper bound of $c^*$, hence we prefer to maximize $\overline{lb}(D_{i+1}, \Omega_{i+1}, u)$ instead of $\overline{lb}(D, \Omega_{i+1}, u)$. However, we do not want to spend too much time finding the dual vector $u$ that maximizes $\overline{lb}(D_{i+1}, \Omega_{i+1}, u)$ because this vector can be discarded later (see below) and, moreover, a weaker vector is usually enough to decide the status of $ub$ as a bound of $c^*$. This is the reason why `B&CG+F` is stopped once its fixing bound reaches $ub$.

After the execution of `B&CG+F`, `PFA` updates $\delta_{i+1}$ to $\max\{\delta, \delta_{i+1}\}$, where $\delta$ is the time required by the last pricing problem solved by `B&CG+F`. This is not a true measure of how much time is required to solve a pricing problem on $D$ and $\Omega_{i+1}$ because $D_{i+1}$ is used instead. However, for the time being, it is our best approximation when $\delta > \delta_{i+1}$. We remark that `B&CG+F` could compute a dual vector $u$ such that $\overline{lb}(D_{i+1}, \Omega_{i+1}, u) \geq ub$. In this case, $ub$ is a lower bound of $c^*$, thus `PFA` stops the fix-augment-and-price and the dual vector $u$ is obtained.

If $\Omega_{i+1} \neq \Omega_i$ when the fix-augment-and-price loop ends, then we solve the pricing problem defined by $u$ on $D$ and $\Omega_{i+1}$. If $\overline{lb}(D, \Omega_{i+1}, u) > \overline{lb}(D, \Omega_i, u_i)$, then we simply set $u_{i+1} = u$ for the next iteration of the fix-augment-and-price loop, even if we cannot guarantee that $u$ maximizes $\overline{lb}(D, \Omega_{i+1}, u)$. Efficiency is the reason why we avoid the re-execution of a column generation algorithm, as it can be too costly. Thus, we optimistically assume that the improved lower bound $\overline{lb}(D, \Omega_{i+1}, u)$ will be strong enough to increase the lower bound of $c^*$ without the need to run a neighborhood augmentation again. However, if $\overline{lb}(D, \Omega_{i+1}, u) \leq \overline{lb}(D, \Omega_i, u_i)$, then we invoke `B&CG+F` with input $D$, $\Omega_{i+1}$, and $\Psi_{i+1}$ to obtain a dual vector $u_{i+1}$ maximizing $\overline{lb}(D, \Omega_{i+1}, u_{i+1})$ because $\overline{lb}(D, \Omega_{i+1}, u_{i+1})$ should be close to $ub$ at each iteration.

## 8 Computational Results

We performed a series of experiments to evaluate the efficiency of our solver as well as to justify some of the decisions we took for its design. All these experiments were executed on single thread in a Workstation running Ubuntu 22.04.1 with an Intel(R) Core(TM) i7-10700 CPU@2.90GHz having 32GB of physical memory. Each program used for the evaluation of the solver was coded in C++17 and compiled with GCC 11.2.0.

CPLEX 12.9 was invoked to solve LMP within each column generation algorithm, using its default parameters, except for `CPX_PARAM_LPMETHOD` = `CPX_ALG_DUAL` and `CPX_PARAM_THREADS` = 1. Thus, LMP was solved with the dual simplex method on a single thread.

Following Roberti and Ruthmair (2021), we evaluated our solver using test instances proposed by Poikonen et al. (2019) for the FSTSP.[1] While Roberti and Ruthmair (2021) considered only those instances with up to $n = 39$ customers, we ran our experiments on those instances with up to $n = 99$ customers. For every $1 \leq i \leq 10$, Poikonen et al. (2019) defined 25 instances with $n = (10i - 1)$ customers. Each of such instances was created by taking $n + 1$ uniformly distributed locations of a $50 \times 50$ grid at random, the first of which was designated as the depot. For each $\alpha \in \{1, 2, 3\}$, a TSP-D instance was obtained by considering that the drone moves $\alpha$ times faster than the truck and that the times required for the truck and drone to travel between locations $v$ and $w$ are measured by the taxicab and Euclidean distance metrics, respectively. That is, if $(x_v, y_v)$ and $(x_w, y_w)$ are the coordinates of $v$ and $w$, respectively, and $0 \leq k \leq n$, then

$$c^{\mathrm{T}}(v, w, k) = \lfloor |x_v - x_w| + |y_v - y_w| \rfloor, \text{and}$$
$$c^{\mathrm{F}}(v, w, k) = c^{\mathrm{J}}(v, w, k) = \lfloor \alpha^{-1}\sqrt{(x_v - x_w)^2 + (y_v - y_w)^2} \rfloor.$$

For $n \geq 49$, we scaled $x_v$ and $y_v$ by a factor of 100 to increase the optimum value of each TSP-D instance from a range of hundreds to a range of tens of thousands, so that a smaller gap can be closed by rounding. We did not scale the instances with $n \leq 39$ to allow a direct comparison against the algorithm by Roberti and Ruthmair (2021).

Table 2 reports the results obtained by `PFA` and the best configuration of the branch-and-price algorithm by Roberti and Ruthmair (2021) (hereafter called `RR`). Both algorithms were executed with a time limit of one hour using the init-ng = 5 customers closer to each vertex for the initial neighborhoods. Let $D$ be the input transport network and $\Omega$ the family of routes defined by the initial neighborhoods. Each row of Table 2 summarizes the results for a number $n$ of customers and a drone speed $\alpha$, where: #inst indicates the number of TSP-D instances of each class (i.e., 25); Gap% is the average of $100(c^* - lb)/c^*$ for the optimum $c^*$ of $\mathrm{MP}(D, \Omega)$ and its lower bound $lb$ obtained by solving $\mathrm{LMP}(D, \Omega)$; $\mathrm{Time_r}$ is the average time required to compute $lb$; $\mathrm{Opt_r}$ counts the number of instances for which the optimum $lb$ of $\mathrm{LMP}(D, \Omega)$ was obtained; Time is the average time spent to solve $\mathrm{MP}(D, \Omega)$; Opt counts the number of instances solved; and LMP describes the number of times $\mathrm{LMP}(D', \Omega')$ had to be solved for some transport network $D'$ and some family of $ng$-feasible routes $\Omega'$. For columns Gap%, Time, and LMP, we consider only those instances that were optimally solved. Similarly, for column $\mathrm{Time_r}$ we consider those instance on which $\mathrm{LMP}(D, \Omega)$ was solved. For `RR`, we approximate LMP as the number of nodes of the branch-and-bound tree, even though bounded nodes are discarded without solving its corresponding linear program.

Table 2 shows the effectiveness of `PFA`: it was able to solve all the instances with 59 customers and almost half of the instances with 99 customers and $\alpha \in \{2, 3\}$, whereas `RR` failed to solve some instances with $n = 29$ and it could only solve 10/50 instances with $n = 39$ and $\alpha \in \{2, 3\}$. One of the reasons for this improvement is our new definition of $ng$-feasibility, that yields lower values of Gap%. Of course, the new dominance rules help to reach such gaps efficiently. This synergy between the new $ng$-feasibility and the dominance rules can be better observed in the detailed reports generated by the algorithm. Indeed, `PFA` closes many instances with $n = 39$ customers and $\alpha = 1$ within its first column generation (i.e., $\lceil lb \rceil = c^*$), whereas `RR` solves only one of these instances, partly because $lb$ is too far from $c^*$ and partly because their labeling algorithm is too costly.

Column LMP of Table 2 depicts a large difference on the number of times `PFA` solves a linear relaxation of MP when compared to `RR`. This behavior is explained by the Fix Step of `PFA` and, to a lesser extent, by Gap%. Indeed, the Fix Step allows `PFA` to increase the lower bound far beyond the lower bound $\lceil lb \rceil$ obtained by the column generation algorithms. Table 3 shows that the time `PFA` spends on its Fix Step is,

[1]Originally, the instaces were publicly available at `http://stefan-poikonen.net/tspd_instance_data.zip`, but that URL seems to be abandoned, at least, since 2020. So, we took the instances from `https://mario.ruthmair.at/wp-content/uploads/2021/02/tspd-instances.zip` ($n \leq 39$) and `http://www.or.unimore.it/site/home/online-resources/exact-models-for-the-fstsp.html` ($n \geq 49$). The former are the instances used in Roberti and Ruthmair (2021), whereas the latter were made public by Dell'Amico et al. (2022).

| | | | Roberti and Ruthmair (2021) | | | | PFA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $\alpha$ | #inst | Gap% | Time | Opt | LMP | Gap% | Time$_\mathrm{r}$ | Opt$_\mathrm{r}$ | Time | Opt | LMP |
| 9 | 1 | 25 | 1.38 | 0.3 | 25 | 4 | 0.11 | 0.1 | 25 | 0.1 | 25 | 1 |
| | 2 | 25 | 1.44 | 0.3 | 25 | 4 | 0.43 | 0.1 | 25 | 0.1 | 25 | 1.0 |
| | 3 | 25 | 0.62 | 0.3 | 25 | 4 | 0.22 | 0.1 | 25 | 0.1 | 25 | 1 |
| 19 | 1 | 25 | 2.20 | 47.4 | 25 | 40 | 0.00 | 1.8 | 25 | 1.8 | 25 | 1 |
| | 2 | 25 | 2.51 | 20.7 | 25 | 38 | 0.38 | 1.6 | 25 | 1.6 | 25 | 1 |
| | 3 | 25 | 3.30 | 17.6 | 25 | 36 | 1.73 | 1.4 | 25 | 2.2 | 25 | 1.3 |
| 29 | 1 | 25 | 1.89 | 1209.0 | 19 | 50 | 0.10 | 12.7 | 25 | 12.8 | 25 | 1 |
| | 2 | 25 | 3.00 | 554.3 | 24 | 118 | 0.43 | 8.8 | 25 | 9.0 | 25 | 1 |
| | 3 | 25 | 4.06 | 537.0 | 23 | 133 | 2.54 | 7.5 | 25 | 23.8 | 25 | 1.8 |
| 39 | 1 | 25 | 1.66 | 3219.7 | 1 | 7 | 0.29 | 48.5 | 25 | 50.4 | 25 | 1 |
| | 2 | 25 | 1.51 | 2604.4 | 3 | 16 | 0.49 | 29.6 | 25 | 30.4 | 25 | 1 |
| | 3 | 25 | 1.50 | 1414.2 | 7 | 44 | 2.08 | 25.5 | 25 | 41.1 | 25 | 1.7 |
| 49 | 1 | 25 | — | — | — | — | 0.83 | 208.5 | 25 | 245.3 | 25 | 1 |
| | 2 | 25 | — | — | — | — | 0.86 | 91.1 | 25 | 166.8 | 25 | 1.3 |
| | 3 | 25 | — | — | — | — | 1.73 | 69.1 | 25 | 101.7 | 25 | 1.6 |
| 59 | 1 | 25 | — | — | — | — | 1.11 | 529.4 | 25 | 650.0 | 25 | 1.1 |
| | 2 | 25 | — | — | — | — | 1.30 | 187.0 | 25 | 254.5 | 25 | 1.4 |
| | 3 | 25 | — | — | — | — | 2.59 | 156.2 | 25 | 333.2 | 25 | 2.6 |
| 69 | 1 | 25 | — | — | — | — | 1.02 | 1228.3 | 25 | 1467.1 | 22 | 1.1 |
| | 2 | 25 | — | — | — | — | 1.77 | 438.1 | 25 | 704.7 | 24 | 2.2 |
| | 3 | 25 | — | — | — | — | 2.60 | 301.5 | 25 | 637.7 | 23 | 3.1 |
| 79 | 1 | 25 | — | — | — | — | 0.96 | 2268.9 | 24 | 2333.9 | 14 | 1.1 |
| | 2 | 25 | — | — | — | — | 1.90 | 759.4 | 25 | 1277.7 | 23 | 2.3 |
| | 3 | 25 | — | — | — | — | 2.73 | 536.5 | 25 | 1243.4 | 24 | 4.0 |
| 89 | 1 | 25 | — | — | — | — | 0.69 | 3024.2 | 9 | 2924.5 | 4 | 1 |
| | 2 | 25 | — | — | — | — | 1.50 | 1343.8 | 25 | 2224.7 | 20 | 3.2 |
| | 3 | 25 | — | — | — | — | 2.60 | 962.5 | 25 | 1947.2 | 20 | 4 |
| 99 | 1 | 25 | — | — | — | — | — | — | 0 | — | 0 | — |
| | 2 | 25 | — | — | — | — | 1.12 | 2258.0 | 25 | 2750.6 | 13 | 2 |
| | 3 | 25 | — | — | — | — | 1.79 | 1506.3 | 25 | 2220.8 | 10 | 3.1 |

Table 2: Computational results for PFC with init-ng = 5

on average, not greater than a quarter than that required by its Price Step. In general, the times of the different components of the PFA method are well balanced. In Table 3, LP% is the average time consumed by CPLEX to solve the different linear problems, 2-Fix% is the average time spent by the two-pass fixing method within B&CG+F, Price% is the average time required by the labeling algorithm, Total% = LP% + 2-Fix% + Price% sums up the average time used by B&CG+F for the Price Step, and Fix Step% is the average time spent by the iterative fixing method; all the average times are percentages relative to the average total time consumed by the method, among those instances optimally solved.

To further support the conclusions obtained from Table 2, we implemented two branch-and-price (BP) algorithms that use the same branching rules given by Roberti and Ruthmair (2021). For the sake of brevity, we omit the description of the branching rules. The only difference between these algorithms is that one solves the linear relaxation using CG, while the other applies B&CG+F. Table 4 summarizes the obtained results. The meaning of the columns is the same as in Table 2, except for the new columns Lbl$_\mathrm{r}$ that count the number of times the bidirectional labeling algorithm is invoked to solve a pricing problem on the whole input network within the first column generation. The columns Gap%, Time$_\mathrm{r}$, and Opt$_\mathrm{r}$ are omitted for B&CG+F because they coincide with those in Table 2. Moreover, for $n \geq 49$ we only report the columns corresponding to the first column generation to reduce the experimentation time, as most instances reach the time limit.

|  |  |  | Price Step |  |  |  |  |
|---|---|---|---|---|---|---|---|
| $n$ | $\alpha$ | #inst | LP% | 2-Fix% | Price% | Total% | Fix Step% |
| 9 | 1 | 25 | 2.3 | 11.8 | 75.4 | 89.5 | 0.4 |
|  | 2 | 25 | 2.6 | 18.4 | 65.3 | 86.3 | 1.5 |
|  | 3 | 25 | 4.2 | 16.8 | 65.5 | 86.5 | 1.6 |
| 19 | 1 | 25 | 4.0 | 15.6 | 75.1 | 94.7 | 0.0 |
|  | 2 | 25 | 4.5 | 22.2 | 65.9 | 92.6 | 1.2 |
|  | 3 | 25 | 3.7 | 19.9 | 60.7 | 84.3 | 9.5 |
| 29 | 1 | 25 | 4.2 | 11.0 | 81.4 | 96.7 | 0.4 |
|  | 2 | 25 | 5.9 | 18.5 | 69.9 | 94.3 | 1.8 |
|  | 3 | 25 | 4.6 | 20.5 | 57.9 | 83.1 | 13.3 |
| 39 | 1 | 25 | 3.8 | 10.4 | 82.3 | 96.5 | 1.6 |
|  | 2 | 25 | 6.9 | 22.1 | 65.6 | 94.6 | 2.4 |
|  | 3 | 25 | 6.3 | 20.5 | 59.5 | 86.3 | 10.6 |
| 49 | 1 | 25 | 2.5 | 15.2 | 72.2 | 89.9 | 8.9 |
|  | 2 | 25 | 5.7 | 22.9 | 62.6 | 91.2 | 6.6 |
|  | 3 | 25 | 7.2 | 20.5 | 57.3 | 85.0 | 12.4 |
| 59 | 1 | 25 | 1.9 | 15.6 | 70.8 | 88.3 | 10.7 |
|  | 2 | 25 | 6.0 | 21.7 | 59.6 | 87.3 | 10.8 |
|  | 3 | 25 | 6.8 | 21.0 | 53.0 | 80.9 | 16.9 |
| 69 | 1 | 25 | 1.5 | 18.7 | 68.8 | 89.0 | 10.2 |
|  | 2 | 25 | 4.1 | 26.1 | 52.7 | 82.9 | 15.5 |
|  | 3 | 25 | 6.2 | 22.0 | 51.2 | 79.3 | 18.7 |
| 79 | 1 | 25 | 1.7 | 18.0 | 68.3 | 88.1 | 11.2 |
|  | 2 | 25 | 4.3 | 25.6 | 53.7 | 83.5 | 15.1 |
|  | 3 | 25 | 5.1 | 24.3 | 49.0 | 78.5 | 19.6 |
| 89 | 1 | 25 | 2.6 | 11.3 | 78.3 | 92.1 | 7.0 |
|  | 2 | 25 | 4.5 | 25.2 | 52.3 | 81.9 | 16.7 |
|  | 3 | 25 | 5.9 | 22.5 | 50.2 | 78.6 | 19.8 |
| 99 | 1 | 25 | — | — | — | — | — |
|  | 2 | 25 | 5.5 | 22.0 | 57.6 | 85.2 | 13.6 |
|  | 3 | 25 | 8.8 | 22.2 | 55.0 | 85.9 | 12.4 |

Table 3: Relative weight of each method in the total time of `PFA` with init-ng = 5

The fact that `PFA` is superior to the BP algorithms is not a surprise, because we devoted little efforts on the implementation of the latter. In particular, we do not use neighborhood augmentation to increase the lower bound before branching, we do not apply a variable fixing method (that would require good upper bounds), and we use a trivial rule to select the next branching variable (one of the most violated) instead of implementing a strong branching scheme. Hence, the direct comparison against `PFA` is not that meaningful. Nevertheless, Table 4 highlights once again that the new notion of *ng*-feasibility and the dominance rules are quite effective. These, and the bidirectional search, are the only additional features of the BP algorithms when compared against `RR`. Moreover, even though we avoid fancy methods for branching, the number of linear relaxations solved by both methods is huge when compared to `PFA`, indicating once again that the increase on the lower bound obtained by branching is not sufficiently strong when compared to the iterative fixing method.

Another interesting aspect of Table 4 is that it allows a direct comparison of `CG` and `B&CG+F`. As mentioned in Section 6.3, `B&CG+F` obtains better lower bounds and requires less time than `CG`. The fact that `B&CG+F` has a lower Gap% is not translated into fewer branch-and-bound nodes (column LMP), at least for those instances with $n \leq 39$, because the BP algorithms take different branching decisions that blur this fact. Still, the difference in Time$_\text{r}$ for the first execution of both algorithms is notorious. Indeed, `CG` is unable to

|  |  |  | CG |  |  |  |  |  |  | B&CG+F |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $\alpha$ | #inst | Gap% | Time$_\mathrm{r}$ | Opt$_\mathrm{r}$ | Lbl$_\mathrm{r}$ | Time | Opt | LMP | Lbl$_\mathrm{r}$ | Time | Opt | LMP |
| 9 | 1 | 25 | 0.11 | 0.1 | 25 | 9.9 | 0.1 | 25 | 1.3 | 8.7 | 0.1 | 25 | 1.3 |
|  | 2 | 25 | 0.43 | 0.1 | 25 | 10.7 | 0.1 | 25 | 1.8 | 9.1 | 0.1 | 25 | 1.7 |
|  | 3 | 25 | 0.39 | 0.1 | 25 | 10.4 | 0.1 | 25 | 2.0 | 9.0 | 0.1 | 25 | 1.6 |
| 19 | 1 | 25 | 0.00 | 3.5 | 25 | 10.7 | 3.6 | 25 | 1.2 | 2.6 | 1.8 | 25 | 1.0 |
|  | 2 | 25 | 0.38 | 1.9 | 25 | 16.4 | 3.4 | 25 | 3.5 | 6.6 | 2.7 | 25 | 3.2 |
|  | 3 | 25 | 1.86 | 1.5 | 25 | 22.1 | 5.5 | 25 | 10.0 | 10.8 | 6.3 | 25 | 12.4 |
| 29 | 1 | 25 | 0.16 | 39.0 | 25 | 23.4 | 76.7 | 25 | 3.6 | 2.8 | 39.7 | 25 | 3.5 |
|  | 2 | 25 | 0.51 | 15.8 | 25 | 29.0 | 62.9 | 25 | 12.2 | 8.2 | 34.8 | 25 | 9.1 |
|  | 3 | 25 | 2.80 | 9.8 | 25 | 28.8 | 94.2 | 25 | 30.3 | 10.5 | 72.6 | 25 | 31.2 |
| 39 | 1 | 25 | 0.16 | 160.2 | 25 | 31.1 | 333.8 | 22 | 4.6 | 2.2 | 102.2 | 22 | 3.5 |
|  | 2 | 25 | 0.58 | 67.1 | 25 | 46.7 | 312.3 | 25 | 15.3 | 6.8 | 212.4 | 25 | 19.4 |
|  | 3 | 25 | 1.90 | 39.7 | 25 | 47.0 | 613.7 | 23 | 62.4 | 13.0 | 525.7 | 24 | 75.7 |
| 49 | 1 | 25 |  | 701.2 | 25 | 51.2 |  |  |  | 3.8 |  |  |  |
|  | 2 | 25 |  | 203.8 | 25 | 58.7 |  |  |  | 8.7 |  |  |  |
|  | 3 | 25 |  | 130.2 | 25 | 63.2 |  |  |  | 12.9 |  |  |  |
| 59 | 1 | 25 |  | 1786.9 | 24 | 65.8 |  |  |  | 4.9 |  |  |  |
|  | 2 | 25 |  | 552.1 | 25 | 84.6 |  |  |  | 7.1 |  |  |  |
|  | 3 | 25 |  | 304.2 | 25 | 78.9 |  |  |  | 14.0 |  |  |  |
| 69 | 1 | 25 |  | 3069.7 | 8 | 64.0 |  |  |  | 5.1 |  |  |  |
|  | 2 | 25 |  | 1096.8 | 25 | 91.4 |  |  |  | 8.1 |  |  |  |
|  | 3 | 25 |  | 675.5 | 25 | 100.0 |  |  |  | 13.2 |  |  |  |
| 79 | 1 | 25 |  | — | 0 | — |  |  |  | 5.4 |  |  |  |
|  | 2 | 25 |  | 2010.0 | 24 | 109.2 |  |  |  | 8.9 |  |  |  |
|  | 3 | 25 |  | 1132.6 | 25 | 110.9 |  |  |  | 15.4 |  |  |  |
| 89 | 1 | 25 |  | — | 0 | — |  |  |  | 4.2 |  |  |  |
|  | 2 | 25 |  | 2999.9 | 9 | 127.8 |  |  |  | 9.3 |  |  |  |
|  | 3 | 25 |  | 2182.5 | 25 | 131.9 |  |  |  | 15.6 |  |  |  |
| 99 | 1 | 25 |  | — | 0 | — |  |  |  | — |  |  |  |
|  | 2 | 25 |  | — | 0 | — |  |  |  | 8.0 |  |  |  |
|  | 3 | 25 |  | 3239.1 | 11 | 145.1 |  |  |  | 11.1 |  |  |  |

Table 4: Comparison of column generations methods for branch-and-price

solve LMP$(D, \Omega)$ on many instances with $n \in \{59, 69\}$, whereas B&CG+F solves LMP$(D, \Omega)$ sufficiently fast, allowing PFA to optimally solve MP$(D, \Omega)$ on most of such instances. Hence, B&CG+F is key to the success of PFA. Column Lbl$_\mathrm{r}$ partly explains the reason behind the efficiency of B&CG+F: the bidirectional labeling algorithm is executed fewer times on the whole transport network, hence most of the generated columns are obtained solving restricted, and thus easier, instances of the pricing problem.

Inside PFA, both the Price and Fix Steps take small jumps for the fixing bound (0.5%) and the candidate bound (0.25%). These values are motivated by a simple experiment with three steps. Emulating B&CG+F, the first step of the experiment invokes CG with $\rho = 0.9$ to obtain a dual vector $u$, while it records the time $\delta$ required to solve the pricing problem defined by $u$. As in the previous experiments, a time limit of one hour is imposed for this first step. In a second step, the two-pass fixing method is executed with $ub = (1 + x)\overline{lb}(u)$, for $x \in \{1.5\%, 3\%, 6\%\}$, using a time limit of $20\delta$. If successful, the number of non-fixed arcs is also recorded. Finally, in the third step, the pricing problem defined by $u$ is solved again in the obtained network, recording the required time $\delta_x$. Table 5 summarizes the results of this experiment, where Gap$_{\overline{lb}}$% is the average of $100(c^* - \overline{lb}(u))/c^*$ for the optimum $c^*$ of MP$(D, \Omega)$, #exp is number of instances on which the first step finishes within an hour, Time is the average of $\delta$, and VF$_x$ has the times %Fix and Lbl% required by the second and third steps, respectively, and the number of arcs %Arcs of the second step,

| | | | | | | $\text{VF}_{1.5\%}$ | | | $\text{VF}_{3\%}$ | | | $\text{VF}_{6\%}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | $\alpha$ | #inst | #exp | $\text{Gap}_{\overline{lb}}$% | Time | %Fix | %Lbl | %Arcs | %Fix | %Lbl | %Arcs | %Fix | %Lbl | %Arcs | #fix |
| 9 | 1 | 25 | 25 | 2.33 | 0.0 | 21.7 | 10.2 | 0.4 | 28.1 | 11.5 | 1.3 | 43.4 | 15.0 | 6.4 | 25 |
| | 2 | 25 | 25 | 3.97 | 0.0 | 29.8 | 9.1 | 0.8 | 36.2 | 10.6 | 1.6 | 61.3 | 14.0 | 10.2 | 25 |
| | 3 | 25 | 25 | 3.80 | 0.0 | 31.0 | 6.2 | 2.3 | 41.1 | 6.7 | 1.0 | 57.7 | 9.7 | 6.8 | 25 |
| 19 | 1 | 25 | 25 | 3.29 | 0.3 | 6.4 | 3.1 | 0.3 | 13.2 | 5.3 | 2.0 | 55.5 | 13.5 | 13.5 | 25 |
| | 2 | 25 | 25 | 5.08 | 0.1 | 14.5 | 2.2 | 0.8 | 27.5 | 4.5 | 2.4 | 86.8 | 13.3 | 13.2 | 25 |
| | 3 | 25 | 25 | 6.90 | 0.1 | 18.2 | 3.3 | 0.5 | 33.2 | 5.8 | 2.4 | 95.5 | 14.6 | 12.5 | 25 |
| 29 | 1 | 25 | 25 | 2.73 | 1.4 | 6.1 | 1.8 | 0.4 | 20.8 | 5.3 | 4.2 | 117.7 | 19.8 | 26.1 | 25 |
| | 2 | 25 | 25 | 4.38 | 0.4 | 11.1 | 1.7 | 0.4 | 31.4 | 4.9 | 2.8 | 142.7 | 18.4 | 19.9 | 25 |
| | 3 | 25 | 25 | 7.96 | 0.2 | 15.5 | 1.8 | 0.5 | 39.5 | 5.1 | 2.4 | 152.2 | 17.8 | 14.8 | 25 |
| 39 | 1 | 25 | 25 | 2.40 | 4.8 | 5.5 | 2.7 | 0.4 | 27.9 | 8.0 | 8.1 | 204.3 | 29.2 | 41.7 | 25 |
| | 2 | 25 | 25 | 4.71 | 1.1 | 12.9 | 1.5 | 0.4 | 45.7 | 5.1 | 3.3 | 229.0 | 21.5 | 23.3 | 25 |
| | 3 | 25 | 25 | 6.83 | 0.6 | 20.1 | 1.7 | 0.5 | 61.3 | 5.6 | 3.1 | 251.1 | 21.0 | 18.5 | 25 |
| 49 | 1 | 25 | 25 | 4.06 | 11.6 | 8.7 | 2.0 | 1.1 | 53.0 | 9.3 | 13.8 | 382.1 | 34.8 | 50.6 | 25 |
| | 2 | 25 | 25 | 4.43 | 2.8 | 16.8 | 1.7 | 0.7 | 75.3 | 7.2 | 6.1 | 394.8 | 29.3 | 34.3 | 25 |
| | 3 | 25 | 25 | 6.64 | 1.5 | 22.9 | 1.7 | 0.5 | 90.8 | 7.1 | 4.8 | 397.4 | 27.4 | 25.1 | 25 |
| 59 | 1 | 25 | 25 | 3.80 | 24.0 | 13.5 | 2.6 | 2.2 | 92.0 | 13.3 | 23.6 | 736.0 | 43.1 | 59.3 | 25 |
| | 2 | 25 | 25 | 4.72 | 5.1 | 22.5 | 1.9 | 0.9 | 113.2 | 8.9 | 8.8 | 577.3 | 33.9 | 35.8 | 25 |
| | 3 | 25 | 25 | 6.85 | 2.9 | 28.4 | 1.8 | 0.6 | 123.3 | 8.6 | 5.9 | 546.8 | 32.3 | 26.9 | 25 |
| 69 | 1 | 25 | 25 | 3.94 | 46.2 | 13.9 | 2.6 | 2.3 | 106.6 | 14.7 | 24.0 | 996.4 | 46.5 | 62.9 | 25 |
| | 2 | 25 | 25 | 4.86 | 9.0 | 30.5 | 2.0 | 1.0 | 152.7 | 10.2 | 10.4 | 803.3 | 37.3 | 40.2 | 25 |
| | 3 | 25 | 25 | 5.48 | 4.9 | 44.3 | 2.2 | 0.9 | 181.8 | 10.3 | 7.9 | 769.0 | 35.3 | 30.3 | 25 |
| 79 | 1 | 25 | 25 | 3.73 | 67.7 | 22.6 | 3.1 | 3.4 | 154.5 | 16.1 | 25.2 | 1194.4 | 44.8 | 58.2 | 17 |
| | 2 | 25 | 25 | 4.78 | 13.6 | 37.7 | 2.1 | 1.3 | 183.1 | 10.7 | 10.7 | 1036.6 | 39.1 | 39.5 | 25 |
| | 3 | 25 | 25 | 5.83 | 7.2 | 45.5 | 2.0 | 0.8 | 205.1 | 10.5 | 7.3 | 944.2 | 37.0 | 29.2 | 25 |
| 89 | 1 | 25 | 25 | 2.57 | 107.2 | 31.3 | 3.8 | 4.9 | 217.2 | 19.2 | 30.2 | 1512.4 | 50.0 | 63.8 | 7 |
| | 2 | 25 | 25 | 5.00 | 20.6 | 48.1 | 2.4 | 1.7 | 235.6 | 12.5 | 12.9 | 1340.5 | 41.7 | 39.9 | 22 |
| | 3 | 25 | 25 | 5.89 | 11.7 | 53.4 | 2.2 | 0.9 | 234.6 | 11.1 | 8.1 | 1140.9 | 38.7 | 30.5 | 25 |
| 99 | 1 | 25 | 23 | — | 169.3 | 38.1 | 4.9 | 8.3 | 267.6 | 21.7 | 33.6 | 1013.0 | 38.9 | 49.0 | 5 |
| | 2 | 25 | 25 | 4.25 | 33.4 | 52.1 | 2.8 | 1.9 | 251.7 | 13.7 | 13.7 | 1329.4 | 40.4 | 38.9 | 16 |
| | 3 | 25 | 25 | 6.37 | 17.3 | 52.7 | 2.2 | 0.9 | 243.5 | 11.5 | 7.8 | 1270.4 | 40.0 | 30.6 | 24 |

Table 5: Bidirectional labeling and two-pass variable fixing

for $x \in \{1.5\%, 3\%, 6\%\}$. The columns Fix% and Price% are percentages relative to Time, whereas %Arcs is a percentage relative to the number of arcs of the input network $D$. All these columns are restricted to those instances for which the second step of the experiment required less than $20\delta$ seconds. Only for $x = 6\%$ there are instances reaching this time limit: column #fix counts the number of instances that did not reach this limit. As usual, $\text{Gap}_{\overline{lb}}$% is restricted to those instances where $\text{MP}(D, \Omega)$ was optimally solved.

As expected, Table 5 shows that the fixing method requires more time and it fixes fewer arcs as the gap increases. More interesting is to observe that the method losses its effectiveness for networks with more customers, for a given $x$. For the hardest cases ($n \geq 59$), a gap of 3% is high enough to make the fixing method harder than the pricing step. Hence, the the small step of 0.5% for the Price Step and the even smaller gap of 0.25% for the Fixing Step, as after each iteration we get even harder problems. Moreover, for the hardest cases, we cannot take much advantage of an upper bound $ub$ of $c^*$, even if $ub$ is tight and cheap to compute (which is not). Indeed, many instances with $n \geq 60$ have a $\text{Gap}_{\overline{lb}}$% of $\geq 6\%$, requiring more than $10\delta$ time, on average, to run (only once) the fixing method to obtain a small speedup. This is a strong reason why we spare the effort of computing tight upper bounds of $c^*$ and, instead, we impose different artificial upper bounds to fasten the execution of `PFA`.

Finally, from Table 5 we can also observe that the pricing problems are harder when $\alpha = 1$, while the obtained lower bound is tighter. This effect is explained by our notion of $ng$-feasibility. It is more common for the truck to visit alone a large sequence of customers $v_1, \ldots, v_k$ when $\alpha = 1$ (i.e., the truck and the drone travel at the same speed) than when $\alpha = 3$ (i.e., the truck is three times slower than the drone). Recall that, within the labeling algorithm, the drone cannot visit $v_1, \ldots, v_k$ when it finally flies to visit a customer. Hence, each move taken by the drone alone is more restricted when $\alpha$ is smaller, reducing the changes for dominance. Consequently, the pricing bound is tighter, but the pricing problem is harder. Ultimately, this effect prevented the `PFA` method to solve any instance with $n = 99$ and $\alpha = 1$ within the imposed time limits.

## 9 Conclusions and Future Research

In this paper we devised a new algorithm to exactly solve the TSP-D in its basic setting. Our labeling algorithm builds upon the one proposed by Roberti and Ruthmair (2021) by strengthening the notion of $ng$-feasibility and dominance rules, and it enumerates the routes using a bidirectional search. When used to solve the pricing problems inside the branch-and-price algorithm proposed by Roberti and Ruthmair (2021), our labeling algorithm is able to find several new solutions with up to $n = 39$ customers (and many instances with $n = 49$, according to preliminary and unreported experiments). Still, the labeling algorithm is too expensive for networks with 59 or more customers. One of the main reason behind this inefficiency is that each time a pricing problem is solved, the labeling algorithm explores a huge number of partial routes that traverse non-promising arcs. To address this issue, we implemented a variable fixing method to discard as many arcs as possible, under the assumption that a valid upper bound $ub$ is known. Then, we developed solver for the TSP-D, called price-fix-and-augment, that applies a DSSR with three main steps: column generation, iterative variable fixing by guessing an upper bound, and neighborhood augmentation. Also, we developed a tailored column generation algorithm that applies variable fixing by guessing different upper bounds, aimed to run the labeling algorithm on the whole network as few times as possible. Experimentation showed that the proposed framework is flexible enough to solve all the instances with $n = 59$ and many instances with up to $n = 99$.

The basic TSP-D is, in some way, an analogous of the traveling salesman problem (TSP) for truck-only problems. Whereas the TSP asks in which order the customers should be visited by a truck, TSP-D asks in which order the customers should be visited by a truck carrying a drone. In real-life TSP problems, the truck has additional operational constraints that complicate the developed algorithms. Still, the research on the basic TSP provides insights that are helpful for many of its variants. Similarly, most of the TSP-D problems proposed thus far have different operational constraints that focus on the technical limitations of the drone. The basic TSP-D helps us to understand how to synchronize the truck and the drone, disregarding other important practical issues. As discussed in Roberti and Ruthmair (2021), many of these operational constraints can be included into the labeling algorithm with not much effort, although others require more attention. For instance, different problems limit the set of customers that can be visited by the drone (constraint $c_1$ in Section 1). It is trivial to incorporate such a constraint in our labeling algorithm, as it suffices to restrict $F^{\mathrm{D}}$ at each step. Similarly, it is not hard to allow *loops* in which the drone visits a customer $v$ while the truck remains stationary at a customer $w$ (constraint $c_6$ in Section 1). Indeed, it suffices to extend the notion of move $(v, x)$ to allow a move with $x = \ell$ where the drone performs the loop. However, even this simple adaptation raises some questions on how to avoid the enumeration of symmetric states within the labeling algorithm. Harder constraints, such as limiting the flying range of the drone, may require revised dominance rules, with its corresponding proofs of correctness, as well as new proofs about the completion bounds and the variable fixing method. It would be interesting, therefore, to consider these operational constraints in future works.

**Acknowledgments.** We are truly grateful to Mario Ruthmair for sharing the benchmark instances, and its optimal values, at an early stage of this project, and to Juan José Miranda Bront for all his support throughout the development of this work.

## References

N. Agatz, P. Bouman, and M. Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transp. Sci.*, 52(4):965–981, 2018. doi:10.1287/trsc.2017.0791.

I. Bakir and G. Ö. Tiniç. Optimizing drone-assisted last-mile deliveries: The vehicle routing problem with flexible drones. *Optim. Online*, pages 1–28, 2020. URL `https://optimization-online.org/?p=16382`. accessed, 11/16/2020.

R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.*, 59(5):1269–1283, 2011. doi:10.1287/opre.1110.0975.

R. Baldacci, A. Mingozzi, and R. Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS J. Comput.*, 24(3):356–371, 2012. doi:10.1287/ijoc.1110.0456.

M. Boccia, A. Masone, A. Sforza, and C. Sterle. An exact approach for a variant of the fs-tsp. *Transp. Res. Proc.*, 52:51–58, 2021a. doi:https://doi.org/10.1016/j.trpro.2021.01.008.

M. Boccia, A. Masone, A. Sforza, and C. Sterle. A column-and-row generation approach for the flying sidekick travelling salesman problem. *Transp. Res. Part C Emerg. Technol.*, 124:102913, 2021b. doi:https://doi.org/10.1016/j.trc.2020.102913.

N. Boland, J. Dethridge, and I. Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Oper. Res. Lett.*, 34(1):58–68, 2006. doi:10.1016/j.orl.2004.11.011.

P. Bouman, N. Agatz, and M. Schmidt. Dynamic programming approaches for the traveling salesman problem with drone. *Networks*, 72(4):528–542, 2018. doi:10.1002/net.21864.

S. Cavani, M. Iori, and R. Roberti. Exact methods for the traveling salesman problem with multiple drones. *Transp. Res. Part C Emerg. Technol.*, 130:103280, 2021. doi:https://doi.org/10.1016/j.trc.2021.103280.

N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981. doi:10.1002/net.3230110207.

S. H. Chung, B. Sah, and J. Lee. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Comput. Oper. Res.*, 123:105004, 2020. doi:10.1016/j.cor.2020.105004.

M. Dell'Amico, R. Montemanni, and S. Novellani. Benchmark instances and optimal solutions for the traveling salesman problem with drone. *CoRR*, abs/2107.13275, 2021a. doi:10.48550/arXiv.2107.13275.

M. Dell'Amico, R. Montemanni, and S. Novellani. Drone-assisted deliveries: new formulations for the flying sidekick traveling salesman problem. *Optim. Lett.*, 15(5):1617–1648, 2021b. doi:10.1007/s11590-019-01492-z.

M. Dell'Amico, R. Montemanni, and S. Novellani. Exact models for the flying sidekick traveling salesman problem. *Int. Trans. Oper. Res.*, 29(3):1360–1393, 2022. doi:10.1111/itor.13030.

M. Dell'Amico, R. Montemanni, and S. Novellani. Algorithms based on branch and bound for the flying sidekick traveling salesman problem. *Omega*, 104:102493, 2021. doi:https://doi.org/10.1016/j.omega.2021.102493.

M. Drexl. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. *Transp. Sci.*, 46(3):297–316, 2012. doi:10.1287/trsc.1110.0400.

E. Es Yurek and H. C. Ozmutlu. A decomposition-based iterative optimization algorithm for traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.*, 91:249–262, 2018. doi:https://doi.org/10.1016/j.trc.2018.04.009.

C. Gambella, A. Lodi, and D. Vigo. Exact solutions for the carrier–vehicle traveling salesman problem. *Transp. Sci.*, 52(2):320–330, 2018. doi:10.1287/trsc.2017.0771.

Q. M. Ha, Y. Deville, Q. D. Pham, and M. H. Hà. On the min-cost traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.*, 86:597–621, 2018. doi:10.1016/j.trc.2017.11.015.

H. Y. Jeong, B. D. Song, and S. Lee. Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones. *Int J. Prod. Econ.*, 214:220–233, 2019. doi:https://doi.org/10.1016/j.ijpe.2019.01.010.

G. Lera-Romero, J. J. Miranda Bront, and F. J. Soulignac. Linear edge costs and labeling algorithms: the case of the time-dependent vehicle routing problem with time windows. *Networks*, 76(1):24–53, 2020. doi:10.1002/net.21937.

G. Lera-Romero, J. J. Miranda Bront, and F. J. Soulignac. Dynamic programming for the time-dependent traveling salesman problem with time windows. *INFORMS J. Comput.*, pages 1–17, 2022. doi:10.1287/ijoc.2022.1236. in press.

Y.-J. Liang and Z.-X. Luo. A survey of truck–drone routing problem: Literature review and research prospects. *J. Oper. Res. Soc. China*, 10(2):343–377, 2022. doi:10.1007/s40305-021-00383-4.

G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and G. Laporte. Drone-aided routing: A literature review. *Transp. Res. Part C Emerg. Technol.*, 120:102762, 2020. doi:10.1016/j.trc.2020.102762.

A. Masone, S. Poikonen, and B. L. Golden. The multivisit drone routing problem with edge launches: an iterative approach with discrete and continuous improvements. *Networks*, 80(2):193–215, 2022. doi:10.1002/net.22087.

M. Moshref-Javadi and M. Winkenbach. Applications and research avenues for drone-based models in logistics: A classification and review. *Expert Syst. Appl.*, 177:114854, 2021. doi:10.1016/j.eswa.2021.114854.

C. C. Murray and A. G. Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transp. Res. Part C Emerg. Technol.*, 54:86–109, 2015. doi:10.1016/j.trc.2015.03.005.

C. C. Murray and R. Raj. The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones. *Transp. Res. Part C Emerg. Technol.*, 110:368–398, 2020. doi:10.1016/j.trc.2019.11.003.

A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: a survey. *Networks*, 72(4):411–458, 2018. doi:10.1002/net.21818.

S. Poikonen and B. Golden. Multi-visit drone routing problem. *Comput. Oper. Res.*, 113:104802, 10, 2020. doi:10.1016/j.cor.2019.104802.

S. Poikonen, X. Wang, and B. Golden. The vehicle routing problem with drones: extended models and connections. *Networks*, 70(1):34–43, 2017. doi:10.1002/net.21746.

S. Poikonen, B. Golden, and E. A. Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS J. Comput.*, 31(2):335–346, 2019. doi:10.1287/ijoc.2018.0826.

A. Ponza. Optimization of drone-assisted parcel delivery. Master's thesis, Universitá Degli Studi di Padova, 2016. URL `https://thesis.unipd.it/handle/20.500.12608/25563`. Accessed 11/17/2022.

G. Righini and M. Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optim.*, 3(3):255–273, 2006. doi:10.1016/j.disopt.2006.05.007.

G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170, 2008. doi:10.1002/net.20212.

R. Roberti and M. Ruthmair. Exact methods for the traveling salesman problem with drone. *Transp. Sci.*, 55(2):315–335, 2021. doi:10.1287/trsc.2020.1017.

D. Schermer, M. Moeini, and O. Wendt. A branch-and-cut approach and alternative formulations for the traveling salesman problem with drone. *Networks*, 76(2):164–186, 2020. doi:10.1002/net.21958.

Z. Tang, W.-J. v. Hoeve, and P. Shaw. A study on the traveling salesman problem with a drone. In L.-M. Rousseau and K. Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 557–564, Cham, 2019. Springer International Publishing.

C. Tilk and S. Irnich. Dynamic programming for the minimum tour duration problem. *Transp. Sci.*, 51(2): 549–565, 2017. doi:10.1287/trsc.2015.0626.

E. van Dijck and P. Bouman. A branch-and-cut algorithm for the traveling salesman problem with drone. Master's thesis, Erasmus University Rotterdam, 2018. URL `http://hdl.handle.net/2105/44107`.

S. A. Vásquez, G. Angulo, and M. A. Klapp. An exact solution method for the TSP with drone based on decomposition. *Comput. Oper. Res.*, 127:Paper No. 105127, 12, 2021. doi:10.1016/j.cor.2020.105127.

X. Wang, S. Poikonen, and B. Golden. The vehicle routing problem with drones: several worst-case results. *Optim. Lett.*, 11(4):679–697, 2017. doi:10.1007/s11590-016-1035-3.

Z. Wang and J.-B. Sheu. Vehicle routing problem with drones. *Transp. Res. Part B Methodol.*, 122:350–364, 2019. doi:10.1016/j.trb.2019.03.005.

## A Proof of Dominance Rules

This section contains the proof of Rules 2–3; the proof of Rule 5 is omitted as is similar to those of Rules 4 and 5.

**Dominance Rule 2.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is inside the truck. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, and $\bar{c}(p) \leq \bar{c}(q)$, then $p$ dominates $q$.*

*Proof.* Since the drone is inside the truck, it follows that $\tau(p) = \tau(q) = 0$ and $F^{\mathrm{D}}(p) = F^{\mathrm{T}}(p) \cup \{t(p)\} \subseteq F^{\mathrm{T}}(q) \cup \{t(q)\} = F^{\mathrm{D}}(q)$. Hence, $p$ dominates $q$ by Rule 1. □

**Dominance Rule 3.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $w \notin F^{\mathrm{D}}(p)$, $\bar{\tau}(p) \leq \bar{\tau}(q)$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), w) \leq \bar{c}(q) + c^{\mathrm{F}}(d(q), w)$, then $p$ dominates $q$ via $w$.*

*Proof.* Suppose $r_q = q + q_1 + q_2$ is an *ng*-feasible route for some sequence of moves $q_1 = (v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d})$. To prove that $p$ dominates $q$ via $w$ (Definition 3), we ought to see that $r_p = p + q_1 + q_2$ is a route, that is *ng*-feasible, and that $\bar{c}(r_p) \leq \bar{c}(r_q)$. Certainly, $r_p$ starts at the depot because so does $p$. Moreover, $r_p$ is a well defined partial route because $r_q$ is a route, $t(p) = t(q)$, and $w \neq d(p)$ because $w \notin F^{\mathrm{D}}(p)$. By definition, the drone is inside the truck in both $q + q_1$ and $p + q_1$. Hence, by induction, $r_p$ ends at the depot because so does $r_q$. Moreover, since $|p| = |q|$, we obtain that $|r_p| = |r_q| = n + 2$. Summing up, $r_p$ is a route.

Regarding the *ng*-feasibility of $r_p$, recall that $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$ by hypothesis. Then, by induction on $|q'|$, (a) $F^{\mathrm{T}}(p + q') \subseteq F^{\mathrm{T}}(q + q')$ for every sequence of moves $q'$. In particular, as the drone is inside the truck in both $p + q_1$ and $q + q_1$, we obtain that $F^{\mathrm{D}}(p + q_1) = F^{\mathrm{T}}(p + q_1) \cup \{v_k\} \subseteq F^{\mathrm{T}}(q + q_1) \cup \{v_k\} = F^{\mathrm{D}}(q + q_1)$. Then, another induction on $|q'|$ is enough to conclude that (b) $F^{\mathrm{D}}(p + q_1 + q') \subseteq F^{\mathrm{D}}(q + q_1 + q')$ for every sequence of moves $q'$. Finally, observe that $w \notin \{v_1, \ldots, v_k\}$ because $q + q_1$ is *ng*-feasible, while $w \notin F^{\mathrm{D}}(p)$ by hypothesis. Therefore, (c) $w \notin F^{\mathrm{D}}(p + q_1)$. Altogether, (a)–(c) imply that $r_p$ is *ng*-feasible.[2]

[2]Observe that the rule is false when Definition 1 is restricted to those routes in which the truck never moves alone to the vertex of the drone, because nothing prevents the truck to visit $d(p)$ within $q + q_1$ when $d(q) \neq d(p)$.

To end the proof we must check that $\bar{c}(r_p) \leq \bar{c}(r_q)$. Clearly, $\tau = \sum_{i=0}^{k-1} c^{\mathrm{T}}(v_i v_{i+1})$ is the time consumed by the truck to go from $v_0 = t(p) = t(q)$ to $v_k$ in $q_1$. Similarly, $u = u_w + \sum_{i=1}^{k} u_{v_i}$ is the sum of the dual values corresponding to the vertices traversed by $q_1$, while $\bar{c}_2 = \bar{c}((v_k, \mathrm{d}) + q_2)$ is the $u$-cost of the partial route traversed after both $p + q_1$ and $q + q_1$. Therefore

$$\begin{aligned}
\bar{c}(r_p) &= \bar{c}(p) + \max\left\{\tau(p) + \tau, c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, v_k)\right\} - u + \bar{c}_2 \\
&= \max\left\{\bar{\tau}(p) + \tau, \bar{c}(p) + c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, v_k)\right\} - u + \bar{c}_2 \\
\text{(hypothesis)} &\leq \max\left\{\bar{\tau}(q) + \tau, \bar{c}(q) + c^{\mathrm{F}}(d(q), w) + c^{\mathrm{J}}(w, v_k)\right\} - u + \bar{c}_2 \\
&= \bar{c}(q) + \max\left\{\tau(q) + \tau, c^{\mathrm{F}}(d(q), w) + c^{\mathrm{J}}(w, v_k)\right\} - u + \bar{c}_2 = \bar{c}(r_q).
\end{aligned}$$

□

**Dominance Rule 4.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $w \notin F^{\mathrm{D}}(p)$, $\bar{\tau}(p) \leq \bar{\tau}(q)$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, t(p)) \leq \bar{\tau}(q)$, then $p$ forking to $w$ dominates $q$ via $w$.*

*Proof.* Suppose $r_q = q + q_1 + q_2$ is an *ng*-feasible route for some sequence of moves $q_1 = (v_1, \mathrm{t})$, …, $(v_k, \mathrm{t})$, $(w, \mathrm{d})$, and let $p_1 = (w, \mathrm{d})$, $(v_1, \mathrm{d})$, …, $(v_k, \mathrm{d})$. To prove that $p$ forking to $w$ dominates $q$ via $w$, we must see that $r_p = p + p_1 + q_2$ is an *ng*-feasible route with $\bar{c}(r_p) \leq \bar{c}(r_q)$. The proof that $r_p$ is an *ng*-feasible route is similar to that in Rule 3 and we omit it for the sake of brevity. Regarding $\bar{c}(r_p) \leq \bar{c}(r_q)$, observe again that $\tau = \sum_{i=0}^{k} c^{\mathrm{T}}(v_i v_{i+1})$ is the time consumed by the truck to go from $v_0 = t(p) = t(q)$ to $v_k$ in both $p_1$ and $q_1$. Similarly, $u = u_w + \sum_{i=1}^{k} u_{v_i}$ is the sum of the dual values corresponding to the vertices traversed by both $p_1$ and $q_1$, while $\bar{c}_2 = \bar{c}((v_k, \mathrm{d}) + q)$ is the $u$-cost of the partial route traversed after $p + p_1$ and $q + q_1$. Therefore

$$\begin{aligned}
\bar{c}(r_p) &= \bar{c}(p) + \max\left\{\tau(p), c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, v_0)\right\} + \tau - u + \bar{c}_2 \\
&= \max\left\{\bar{\tau}(p), \bar{c}(p) + c^{\mathrm{F}}(d(p), w) + c^{\mathrm{J}}(w, v_0)\right\} + \tau - u + \bar{c}_2 \\
\text{(hypothesis)} &\leq \bar{\tau}(q) + \tau - u + \bar{c}_2 = \bar{c}(q) + \tau(q) + \tau - u + \bar{c}_2 \\
&\leq \bar{c}(q) + \max\left\{\tau(q) + \tau, c^{\mathrm{F}}(d(q), w) + c^{\mathrm{J}}(w, v_k)\right\} - u + \bar{c}_2 = \bar{c}(r_q).
\end{aligned}$$

□

**Dominance Rule 5.** *Let $p$ and $q$ be ng-feasible partial routes in which the drone is outside the truck, $v \in V_c$, and $w \in V_c \setminus F^{\mathrm{D}}(q)$. If $|p| = |q|$, $t(p) = t(q)$, $F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(q)$, $v \notin F^{\mathrm{D}}(p) \cup N(t(p))$, $\bar{\tau}(p) - u_v \leq \bar{\tau}(q) - u_w$, and $\bar{c}(p) + c^{\mathrm{F}}(d(p), v) + c^{\mathrm{J}}(v, t(p)) - u_v \leq \bar{\tau}(q) - u_w$, then $p$ forking to $v$ dominates $q$ via $w$.*

*Proof.* We only sketch the proof, as it is similar to that of Rule 4. As before, we must prove that $r_p = p + p_1 + p_2$ is an *ng*-feasible route with $\bar{c}(r_p) \leq \bar{c}(r_q)$, for every *ng*-feasible route $r_q = q + q_1 + q_2$ such that $q_1 = (v_1, \mathrm{t}), \ldots, (v_k, \mathrm{t}), (w, \mathrm{d})$, where $p_1 = (v, \mathrm{d})$, $(v_1, \mathrm{d})$, …, $(v_k, \mathrm{d})$. The proof that $r_p$ is a route is analogous to that in Rule 3, thus we omit it. That $\bar{c}(r_p) \leq \bar{c}(r_q)$ also follows as in Rule 4; just replace $w$ with $v$ for the arithmetic operations involving $p$ (such a replacement also affects the sum of the dual values). Finally, regarding the *ng*-feasibility of $r_p$, observe that $p + (v, \mathrm{d})$ is *ng*-feasible because $v \notin F^{\mathrm{D}}(p)$. Moreover, $F^{\mathrm{T}}(p + (v, \mathrm{d})) = F^{\mathrm{T}}(p)$ because $v \notin N(t(p))$. Then, by induction on $i$, it follows that $F^{\mathrm{T}}(p + (v, \mathrm{d}), (v_1, \mathrm{d}), \ldots, (v_i, \mathrm{d})) \subseteq F^{\mathrm{T}}(q + (v_1, \mathrm{t}), \ldots, (v_i, \mathrm{t}))$ for every $1 \leq i \leq k$. In particular, $F^{\mathrm{T}}(p + p_1) \subseteq F^{\mathrm{T}}(q + (q_1 - (w, \mathrm{d}))) \subseteq F^{\mathrm{T}}(q + q_1)$. Then, by induction on $|q'|$, we obtain that $F^{\mathrm{T}}(p + p_1 + q') \subseteq F^{\mathrm{T}}(q + q_1 + q')$ for every sequence of moves $q'$. Altogether, these conditions imply that $r_q$ is *ng*-feasible, as desired. □

## B Correctness of the Bidirectional Search

This section contains the proofs of Theorem 1 and 2.

**Theorem 1.** *The following conditions are equivalent for a route $r$ of $D$.*

1. *$r$ is ng-feasible,*

2. *$r$ is ng-feasible at some $1 \leq m \leq n+2$, and*

3. *$r$ is ng-feasible at every $1 \leq m \leq n+2$.*

*Proof.* 1 $\Rightarrow$ 2 and 3 $\Rightarrow$ 1 are trivial. For 2 $\Rightarrow$ 3, we first prove that 2 implies 4: if $m > 1$, then $r$ is *ng*-feasible at $m-1$. For this, suppose $r = p \oplus (v, x) \oplus q$ for $|p| = m$, and consider the following cases.

**Case 1:** $x = \mathrm{t}$, thus $t(p) = t(q)$ and the drone is inside the truck in both $p$ and $q$. Hence, $p = p' + (w, \mathrm{d})$ for some partial route $p'$ and some vertex $w$. We divide this case into two alternatives.

**Case 1.1:** the truck carries the drone from $v' = d(p')$ to $v = w$ in the last move of $p$. In this case, $r = p' \oplus q'$ for the partial route $q' = q + (v', \mathrm{d})$ of $D^{-1}$. Certainly, $p'$ is *ng*-feasible by definition. Moreover, either $v' \notin N(v)$ or $v' \in F^{\mathrm{T}}(p) = (F^{\mathrm{T}}(p') \cup \{v'\}) \cap N(v)$. In the former case, $v' \notin F^{\mathrm{T}}(q)$ because $F^{\mathrm{T}}(q) \subseteq N(v)$; in the latter case, $v' \notin F^{\mathrm{T}}(q)$ because $r$ is *ng*-feasible at $m$, thus $F^{\mathrm{T}}(p) \cap F^{\mathrm{T}}(q) = \emptyset$. Consequently, $q'$ is *ng*-feasible. Finally, suppose $z \in F^{\mathrm{T}}(q') \subseteq N(v')$ and observe that $z \neq v'$ because $v' \notin F^{\mathrm{T}}(q) \subseteq F^{\mathrm{T}}(q') \cup \{v\}$. If $z = v$, then $z \notin F^{\mathrm{T}}(p')$ because $p = p' + (v, \mathrm{d})$ is *ng*-feasible. Otherwise, $z \in F^{\mathrm{T}}(q) \supseteq F^{\mathrm{T}}(q') \setminus \{v\}$ and, therefore, $z \notin F^{\mathrm{T}}(p) \subseteq F^{\mathrm{T}}(p') \cup \{v\}$ because $r$ is *ng*-feasible at $m$. In both cases, $z \notin F^{\mathrm{T}}(p')$, thus $F^{\mathrm{T}}(p') \cap F^{\mathrm{T}}(q') = \emptyset$. Altogether, $r$ is *ng*-feasible at $m-1$.

**Case 1.2:** the drone departs from $d(p')$ to visit $w \neq v$ alone, and then it meets the truck at $v$. In this case, $r = p' \oplus (w, \mathrm{d}) \oplus q$ and both $p'$ and $q$ are *ng*-feasible by hypothesis. If $z \in F^{\mathrm{T}}(p') \subseteq N(v)$, then $z \in F^{\mathrm{T}}(p) = (F^{\mathrm{T}}(p') \cup \{w\}) \cap N(v)$ and, consequently, $z \notin F^{\mathrm{T}}(q)$ because $r$ is *ng*-feasible at $m$. That is, $F^{\mathrm{T}}(p') \cap F^{\mathrm{T}}(q) = \emptyset$. Finally, $w \notin F^{\mathrm{D}}(p')$ because $p = p' + (w, \mathrm{d})$ is *ng*-feasible. Similarly, either $w \notin N(v)$ or $w \in F^{\mathrm{T}}(p) = (F^{\mathrm{T}}(p') \cup \{w\}) \cap N(v)$. Whichever the case, $w \notin F^{\mathrm{D}}(q) = F^{\mathrm{T}}(q) \cup \{v\} \subseteq N(v) \cup \{v\}$ because $r$ is *ng*-feasible at $m$. Altogether, $r$ is *ng*-feasible at $m-1$.

**Case 2:** $x = \mathrm{d}$, thus the drone is outside the truck in $p$ and, therefore, $p = p' + (w, \mathrm{t})$ for some partial route $p'$ and some vertex $w$. Consider the following possibilities.

**Case 2.1:** the drone is inside the truck in $p'$, thus the truck splits from the drone at $w' = t(p')$ and it travels alone to $w$ in the last move of $p$. In this case, $r = p' \oplus q'$ for the partial route $q' = q + (w', \mathrm{t}) + (v, \mathrm{d})$. The partial route $p'$ is *ng*-feasible by hypothesis. If $w' \in N(w)$, then $w' \in F^{\mathrm{T}}(p) = (F^{\mathrm{T}}(p') \cup \{w'\}) \cap N(w)$ and, therefore, $w' \notin F^{\mathrm{T}}(q) \subseteq N(w)$ because $r$ is *ng*-feasible at $m$. Consequently, $q + (w', \mathrm{t})$ is *ng*-feasible. Similarly, $v \notin F^{\mathrm{D}}(q)$ because $r$ is *ng*-feasible at $m$ and, consequently, $v \notin F^{\mathrm{D}}(q + (w', \mathrm{t})) = F^{\mathrm{D}}(q) \cup \{w'\}$ because $v \neq w' \in F^{\mathrm{D}}(p)$. This means that $q'$ is also *ng*-feasible. Suppose $z \in F^{\mathrm{T}}(q')$. If $z = v$, then $z \notin F^{\mathrm{T}}(p') \subseteq F^{\mathrm{D}}(p') \cup \{w\} = F^{\mathrm{D}}(p)$ because $r$ is *ng*-feasible at $m$. Also, $z \neq w'$ because $w' \notin F^{\mathrm{T}}(q')$. If $z = w$, then $z \notin F^{\mathrm{T}}(p')$ because $p = p' + (w, \mathrm{t})$ is *ng*-feasible. Finally, if $z \notin \{v, w, w'\}$, then $z \in F^{\mathrm{T}}(q) \subseteq N(w)$. Hence, as $r$ is *ng*-feasible at $m$, it follows that $z \notin F^{\mathrm{T}}(p) = (F^{\mathrm{T}}(p') \cup \{w'\}) \cap N(w)$ and, therefore, $z \notin F^{\mathrm{T}}(p')$. Summing up, $F^{\mathrm{T}}(p') \cap F^{\mathrm{T}}(q') = \emptyset$ and, altogether, $r$ is *ng*-feasible at $m-1$.

**Case 2.2:** the drone is outside the truck in $p'$, thus the truck was alone at $w' = t(p')$ when it departed to $w$ in the last move of $p$. In this case, $r = p' \oplus (v, \mathrm{d}) \oplus q'$ for the partial route $q' = q + (w', \mathrm{t})$. We omit the facts that $p'$ and $q'$ are *ng*-feasible and $F^{\mathrm{T}}(p') \cap F^{\mathrm{T}}(q) = \emptyset$ as they follow as in Case 2.1. Finally, as $r$ is *ng*-feasible at $m$, we obtain that $v \notin F^{\mathrm{D}}(p') \cup \{w'\} \subseteq F^{\mathrm{D}}(p)$ and, hence, $v \notin F^{\mathrm{D}}(q') = F^{\mathrm{D}}(q) \cup \{w'\}$. Consequently, $r$ is *ng*-feasible at $m-1$.

To complete the proof of 2 $\Rightarrow$ 3, suppose $r$ is *ng*-feasible at $m$. Applying $m-1$ times the implication 2 $\Rightarrow$ 4, we obtain that $r$ is *ng*-feasible at 1. This means that $q$ is an *ng*-feasible partial route of $D^{-1}$, for the unique $q$ such that $r = (0, \mathrm{d}) \oplus q$. By definition, $q$ is a route of $D^{-1}$. Moreover, viewed in $D^{-1}$, the route is $q = (0, \mathrm{d}) \oplus r$. Then, applying $n+1$ times the impliciation 2 $\Rightarrow$ 4 on $q$ and $D^{-1}$, we obtain that $q$ is *ng*-feasible at 1. Getting back to $D$, we obtain that $r$ is *ng*-feasible. Finally, applying $i$ times the implication 2 $\Rightarrow$ 4, we obtian that $r$ is *ng*-feasible at $n+2-i$ for $0 \leq i \leq n+1$, as desired. ☐

Before dealing with Theorem 2, we prove two technical, yet important results, that explain why a route is discarded by the labeling algorithm. These results are also the basis of completion bounds and the variable fixing method. It is for the latter that we consider an additional configuration of the labeling algorithm in which some dominance rules are disabled. Recall that each partial route $p$ has an associated partial route $\chi(p)$ with the same truck and drone paths, except the truck carries the drone after the bifurcation point of $p$ (Figure 5).

**Proposition 3.** *For $1 \leq m < n+2$, let $\mathcal{P}_m$ be the family of ng-feasible partial routes of length $m$ that is obtained after the labeling algorithm is applied on a transport network $D$. If $r = p \oplus (v, x) \oplus q$ is an ng-feasible route of $D$ and $|p| = m$, then $D$ has an ng-feasible route $r' = p' \oplus (v', x') \oplus q'$ such that $p' \in \mathcal{P}_m$, $\bar{c}(r') \leq \bar{c}(r)$, and:*

1. *if $x = \mathrm{t}$, then $x' = \mathrm{t}$ and $q' = q$,*
2. *if $x = \mathrm{d}$ and $x' = \mathrm{t}$, then $q' = \chi(q) + (v', \mathrm{d})$,*
3. *if $x = x' = \mathrm{d}$, then $q' \in \{q, \chi(q)\}$ and $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p')$,*
4. *if Rules 4–6 are disabled and $x = \mathrm{d}$, then $x' = \mathrm{d}$, $v' = v$, and $q' = q$.*

*Proof.* The proof is by induction on $m$. The base case $m = 1$ is true because $\mathcal{P}_1 = \{(0, \mathrm{d})\}$ and $r = (0, \mathrm{d}) \oplus q$ for every *ng*-feasible route $q$ of $D^{-1}$. For the inductive step $m > 1$, we consider the following alternatives.

**Case 1:** $x = \mathrm{t}$ and $p = \phi + (v, \mathrm{d})$ for some partial route $\phi$. In this case, the truck carries the drone from $w = t(\phi)$ to $t(p) = v$ in the last move of $p$, thus $r = \phi \oplus (q + (w, \mathrm{d}))$. By the inductive hypothesis, $D$ has an *ng*-feasible route $r_0 = \phi' \oplus (q + (w, \mathrm{d}))$ such that $\phi' \in \mathcal{P}_{m-1}$ and $\bar{c}(r_0) \leq \bar{c}(r)$. Note that $v \notin F^{\mathrm{T}}(\phi')$ because $p_0 = \phi' + (v, \mathrm{d})$ is a partial route of the *ng*-feasible route $r_0 = p_0 \oplus q$. Then, at the $(m-1)$-th iteration of the labeling algorithm, $\phi'$ will get extended into $p_0$. Certainly, $\bar{c}(p_0) \leq \bar{c}(p)$ because $\bar{c}(\phi') \leq \bar{c}(\phi)$. Consider the longest sequence $p_0, \ldots, p_k$ such that the labeling algorithm detects that $p_{j+1}$ dominates $p_j$ according to Rule 2, for $0 \leq j < k$. This sequence is well defined because the labeling algorithm stops after a finite number of steps. By Rule 2, the drone is inside the truck in $p_j$ for every $j \geq 0$, thus Rule 2 is the only reason why the labeling algorithm may discard $p_k$. Hence, $p_k \in \mathcal{P}_m$ by the maximality of $k$. Moreover, by Definition 2, $r_j = p_j \oplus q$ is an *ng*-feasible route with $\bar{c}(r_j) \leq \bar{c}(r_{j-1})$ for every $j > 0$, thus $r' = r_k$ satisfies the proposition.

**Case 2:** $x = \mathrm{t}$ and $p = \phi + (w, \mathrm{d})$ for some partial route $\phi$ and $w \neq v$. In other words, in the last move of $p$, the drone leaves $d(\phi)$ to visit $w$, and then it travels to $v$ where it meets the truck again. Thus, $r = \phi \oplus (w, \mathrm{d}) \oplus q$. Since $x = \mathrm{t}$, the drone is inside the truck in $q$. This means that $\chi(q) = q$, hence there are two possibilities by the inductive hypothesis for $m - 1$.

**Case 2.1:** Condition 2 is true, thus $D$ has a route $r_0 = \phi' \oplus (q + (w', \mathrm{d}))$ such that $\phi' \in \mathcal{P}_{m-1}$ and $\bar{c}(r_0) \leq \bar{c}(r)$. As before, the $(m-1)$-th iteration extends $\phi'$ into $p_0 = \phi' + (v, \mathrm{d})$. Since $r_0 = p_0 \oplus q$, the same arguments of Case 1 imply the existence of the desired route $r' = p' \oplus q$ such that $p' \in \mathcal{P}_m$; we omit the details for the sake of brevity.

**Case 2.2:** Condition 3 is true, thus $D$ has a route $r_0 = \phi' \oplus (w', \mathrm{d}) \oplus q$ such that $\phi' \in \mathcal{P}_{m-1}$, $\bar{c}(r_0) \leq \bar{c}(r)$, and $w' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(\phi')$. The last condition implies that the $(m-1)$-th iteration of the labeling algorithm extends $\phi'$ into $p_0 = \phi' + (w', \mathrm{d})$. Hence, the desired route $r' = p' \oplus q$ such that $p' \in \mathcal{P}_m$ is obtained from $r_0 = p_0 \oplus q$ as in Case 1.

**Case 3:** $x = \mathrm{d}$, thus $p = \phi + (w, \mathrm{t})$ for some partial route $\phi$ and $w \neq v$. This time, the truck travels alone from some vertex $z = t(\phi)$ to $t(p) = w$. Consequently, $r = \phi \oplus (v, \mathrm{d}) \oplus (q + (z, \mathrm{t}))$ and we have three possibilities by the inductive hypothesis for $m - 1$.

**Case 3.1:** Condition 2 is true, thus $D$ has a route $r_0 = \phi' \oplus (\chi(q) + (z, \mathrm{d}) + (v', \mathrm{d}))$ such that $\phi' \in \mathcal{P}_{m-1}$ and $\bar{c}(r_0) \leq \bar{c}(r)$. By Condition 4, this case happens only if at least one of the Rules 4–6 is enabled. At the $(m-1)$-th iteration, the labeling algorithm will extend $\phi'$ into $p_0 = \phi' + (z, \mathrm{d})$, from which we can find the desired route $r' = p' \oplus (\chi(q) + (z, \mathrm{d}))$ as in Case 1.

**Case 3.2:** Condition 3 is true for $q' = \chi(q + (z, \mathrm{t})) = \chi(q) + (z, \mathrm{d})$, thus $D$ has a route $r_0 = \phi' \oplus (v', \mathrm{d}) \oplus (\chi(q) + (z, \mathrm{d}))$ such that $\phi' \in \mathcal{P}_{m-1}$, $\bar{c}(r_0) \leq \bar{c}(r)$, and $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(\phi')$. By Condition 4, this case happens only if at least one of the Rules 4–6 is enabled. At the $(m-1)$-th iteration, the algorithm will create the partial route $p_0 = \phi' + (v', \mathrm{d})$ because $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(\phi)$. Certainly, $r_0 = p_0 \oplus (\chi(q) + (z, \mathrm{d}))$, thus we can find the desired route $r' = p' \oplus (\chi(q) + (z, \mathrm{d}))$ as in Case 1.

**Case 3.3:** Condition 3 is true for $q' = q + (z, \mathrm{t})$, thus $D$ has a route $r_0 = \phi' \oplus (v_0, \mathrm{d}) \oplus (q + (z, \mathrm{t}))$ such that $\phi' \in \mathcal{P}_{m-1}$, $\bar{c}(r_0) \leq \bar{c}(r)$ and $v_0 \notin F^{\mathrm{D}}_{\mathrm{lbl}}(\phi')$. Moreover, $v_0 = v$ if Rules 4–6 are disabled. At the $(m-1)$-th iteration of the labeling algorithm, $\phi'$ will be extended into $p_0 = \phi' + (w, \mathrm{t})$ because $w \notin F^{\mathrm{T}}(\phi')$ by the *ng*-feasibility of $r_0 = p_0 \oplus (v_0, \mathrm{d}) \oplus q$. As before, $\bar{c}(p_0) \leq \bar{c}(p)$ follows because $\bar{c}(\phi') \leq \bar{c}(\phi)$. The *ng*-feasibility of $r_0$ implies also that $v_0 \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p_0)$ the moment $p_0$ is created. Later, $v_0$ may be inserted into $F^{\mathrm{D}}_{\mathrm{lbl}}(p_0)$ because of a dominance relation. If $p_0$ is detected to be self-dominated via $v_0$ according to Rule 6, then $p_1 = \phi' + (v_0, \mathrm{d})$ is also generated at the $(m-1)$-th iteration of the labeling algorithm. By Definition 5, $r_1 = p_1 \oplus (\chi(q) + (z, \mathrm{d}))$ is an *ng*-feasible route with $\bar{c}(r_1) \leq \bar{c}(r_0) \leq \bar{c}(r)$. In this case, we can obtain the desired route $r'$ from $r_1$ as in Case 1. Suppose, then, that $p_0$ is not self-dominated by Rule 6, and consider the longest sequence $p_0, \ldots, p_k$ such that the labeling algorithm detects that either $p_{j+1}$ dominates $p_j$ via $v_{j+1} = v_j$ according to Rule 3 or $p_{j+1}$ forking to $v_{j+1}$ dominates $p_j$ via $v_j$ according to Rules 4 or 5. Again, this sequence is well defined because the labeling algorithm runs for a finite amount of time. Certainly, $p_j$ is not self-dominated according to Rule 6 for $j > 0$; otherwise, the algorithm never compares $p_{j-1}$ against $p_j$. Moreover, when $p_{j-1}$ and $p_j$ are compared, we know that $v_j \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p_j)$, and $v_j = v_{j-1}$ if Rule 5 is disabled. Altogether, $p_k \in \mathcal{P}_m$ and $v_k \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p_k)$ by the maximality of $k$. Moreover, by Definitions 3 and 4, $r_j = p_j \oplus (v_j, \mathrm{d}) \oplus q_j$ is an *ng*-feasible route with $\bar{c}(r_j) \leq \bar{c}(r_{j-1})$ for $j > 0$, where $q_j \in \{q_{j-1}, \chi(q_{j-1})\}$, and $q_{j+1} = q_j$ if Rules 4 and 5 are disabled. Summing up, $r' = r_k$ satisfies the proposition.

□

**Proposition 4.** *For $1 \leq m < n + 2$, let $\mathcal{P}_m$ be the family of ng-feasible partial routes of length $m$ that is obtained after the labeling algorithm is applied on the reverse $D^{-1}$ of a transport network $D$. If $r = p \oplus (v, x) \oplus q$ is an ng-feasible route of $D$ and $|q| = m$, then $D$ has an ng-feasible route $r' = p' \oplus (v', x') \oplus q'$ such that $q' \in \mathcal{P}_m$, $\bar{c}(r') \leq \bar{c}(r)$, and:*

1. *if $x = \mathrm{t}$, then $x' = \mathrm{t}$ and $p' = p$,*
2. *if $x = \mathrm{d}$ and $x' = \mathrm{t}$, then $p' = \chi(p) + (v', \mathrm{d})$.*
3. *if $x = x' = \mathrm{d}$, then $p' \in \{p, \chi(p)\}$, $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(q')$, and either $p' = \chi(p)$ or $v' = v$ or the drone is outside the truck in $q'$.*

*Proof.* For the sake of clarity, we divide the proof in two cases.

**Case 1:** $x = \mathrm{d}$ and the drone is inside the truck in $q$. In this case, $r = (p + (v, \mathrm{d})) \oplus q$. Moreover, $r_{-1} = q \oplus (p + (v, \mathrm{d}))$ is equal to the route $(0, \mathrm{d}) \oplus r$ of $D^{-1}$ that corresponds to $r$, whose *ng*-feasibility follows by Corollary 1. Then, by Proposition 3, $D^{-1}$ has an *ng*-feasible route $r'_{-1} = q' \oplus (p + (v, \mathrm{d}))$ such that $\bar{c}(r'_{-1}) \leq \bar{c}(r_{-1}) = \bar{c}(r)$ and $q' \in \mathcal{P}_m$. Route $r'_{-1}$ corresponds to an *ng*-feasible route $r' = (p + (v, \mathrm{d})) \oplus q' = p \oplus (v, \mathrm{d}) \oplus q'$ of $D$ (Corollary 1). As $v \notin F^{\mathrm{D}}(q') = F^{\mathrm{D}}_{\mathrm{lbl}}(q')$ by the *ng*-feasibility of $r'$, it follows that $r'$ satisfies the corollary.

**Case 2:** $x = \mathrm{t}$ or the drone is outside the truck in $q$. This time, $r_{-1} = q \oplus (v, x) \oplus p$ is an *ng*-feasible route of $D^{-1}$ (Corollary 1). By Proposition 3, $D^{-1}$ has an *ng*-feasible route $r'_{-1} = q' \oplus (v', x') \oplus p'$ such that $\bar{c}(r'_{-1}) \leq \bar{c}(r_{-1}) \leq \bar{c}(r)$ and $q' \in \mathcal{P}_m$. Certainly, the drone is outside the truck in $q'$ when $x' = \mathrm{d}$. If $x' = \mathrm{t}$ or the drone is inside the truck in $p'$, then $r'_{-1}$ corresponds to an *ng*-feasible route $r' = p' \oplus (v', x') \oplus q'$ of $D$ (Corollary 1) that satisfies the proposition. Otherwise, $p' \in \{p, \chi(p)\}$, $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(q')$, and $r'_{-1} = (q' + (v', \mathrm{d})) \oplus p'$ corresponds to the *ng*-feasible route $r' = p' \oplus (q' + (v, \mathrm{d})) = p' \oplus (v', \mathrm{d}) \oplus q'$ of $D$ (Corollary 1) that satisfies the proposition.

□

**Theorem 2.** *The family $\mathcal{R}$ computed by the bidirectional search contains at least one ng-feasible route with a minimum u-cost.*

*Proof.* Let $\bar{c}$ be the minimum among the $u$-costs of the *ng*-feasible routes, and $\mathcal{F}$ and $\mathcal{B}$ be the families of partial routes computed by the labeling algorithm at the forward and backward steps, respectively. By Theorem 1, it suffices to see that $\mathcal{R}$ has route with $u$-cost $\bar{c}$ that is *ng*-feasible at $m$. We consider three cases in which $|p| = m$, where Case $i$ assumes that Case $(i-1)$ does not hold for $i \in \{2, 3\}$.

**Case 1:** some route $r = p \oplus q$ has $\bar{c}(r) = \bar{c}$. By Proposition 3, $D$ has an *ng*-feasible route $r' = p' \oplus q$ with $p' \in \mathcal{F}$ and $\bar{c}(r') = \bar{c}$. Then, by Proposition 4, $D$ has an *ng*-feasible route $r^* = p' \oplus q'$ such that $q' \in \mathcal{B}$ and $\bar{c}(r^*) = \bar{c}$. By Theorem 1, $r^*$ is *ng*-feasible at $m$, thus the merge step inserts $r^*$ into $\mathcal{R}$.

**Case 2:** some route $r = p \oplus (v, \mathrm{d}) \oplus q$ in which the drone is inside the truck in $q$ has $\bar{c}(r) = \bar{c}$. Note that $\chi(q) = q$ by definition. Then, as Case 1 does not hold, Proposition 3 implies that $D$ has an *ng*-feasible route $r' = p' \oplus (v', \mathrm{d}) \oplus q$ such that $p' \in \mathcal{F}$, $\bar{c}(r') = \bar{c}$, and $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p')$. Certainly, $r' = (p' + (v', \mathrm{d})) \oplus q$. Then, by Proposition 4, $D$ has an *ng*-feasible route $r^* = (p' + (v', \mathrm{d})) \oplus q'$ such that $\bar{c}(r^*) = \bar{c}$ and $q' \in \mathcal{B}$. Certainly, $v' \notin F^{\mathrm{D}}(q') = F^{\mathrm{D}}_{\mathrm{lbl}}(q')$ because $r^* = p' \oplus (v', \mathrm{d}) \oplus q'$ is *ng*-feasible. By Theorem 1, $r^*$ is *ng*-feasible at $m$, thus the merge step inserts $r^*$ into $\mathcal{R}$.

**Case 3:** some route $r = p \oplus (v, \mathrm{d}) \oplus q$ in which the drone is outside the truck in $q$ has $\bar{c}(r) = \bar{c}$. As Cases 1 and 2 do not hold, Proposition 3 implies that $D$ has an *ng*-feasible route $r' = p' \oplus (w, \mathrm{d}) \oplus q$ such that $p' \in \mathcal{F}$ and $\bar{c}(r') = \bar{c}$. Similarly, as Case 1 does not hold, Proposition 4 implies that $D$ has an *ng*-feasible route $r^* = p' \oplus (z, \mathrm{d}) \oplus q'$ such that $q' \in \mathcal{B}$, $\bar{c}(r') \leq \bar{c}(r)$, and $z \notin F^{\mathrm{D}}_{\mathrm{lbl}}(q')$. As before, $z \notin F^{\mathrm{D}}(p')$ because $r^* = p' \oplus (z, \mathrm{d}) \oplus q'$ is *ng*-feasible. Moreover, by Theorem 1, $r^*$ is *ng*-feasible at $m$, thus the merge step inserts $r^*$ into $\mathcal{R}$.

□

## C Correctness of the Variable Fixing Methods

**Proposition 1.** *Let $\mathcal{B}$ be the family of ng-feasible routes that is obtained by running the labeling algorithm on $D^{-1}$. If $r = p \oplus (v, x) \oplus q$ is an ng-feasible route of $D$, and $v \notin F^{\mathrm{D}}_{\mathrm{lbl}}(p)$ when $x = \mathrm{t}$, then $cb_{\mathcal{B}}(p) \leq \bar{c}(r)$.*

*Proof.* By Proposition 4, $D$ has an *ng*-feasible route $r' = p' \oplus (v', x') \oplus q'$ such that $\bar{c}(r') \leq \bar{c}(r)$ and $q' \in \mathcal{B}$. Consider the following possibilities that arise from Proposition 4.

**Case 1:** $x = \mathrm{t}$, thus $r' = p \oplus q'$. Certainly, $cb(p) \leq \bar{c}(r') \leq \bar{c}(r)$.

**Case 2:** $x = x' = \mathrm{d}$, thus $p' \in \{p, \chi(p)\}$, $v' \notin F^{\mathrm{D}}_{\mathrm{lbl}}(q')$, and either $p = \chi(p)$, or $v' = v$, or the drone is outside the truck in $q'$. If $p' = \chi(p)$, then $r' = \chi(p) \oplus q'$, hence $cb(p) \leq cb(\chi(p)) \leq \bar{c}(r') \leq \bar{c}(r)$. If $p' = p$ and $v' = v$, then $r' = p \oplus (v, \mathrm{d}) \oplus q'$, thus $cb(p) \leq cb_1(p) \leq \bar{c}(r') \leq \bar{c}(r)$. Finally, if $p' = p$ and the drone is outside the truck in $q$, then $r' = p \oplus (v', \mathrm{d}) \oplus q'$, thus $cb(p) \leq cb_2(p) \leq \bar{c}(r') \leq \bar{c}(r)$.

**Case 3:** $x = \mathrm{d}$ and $x' = \mathrm{t}$, thus $r' = (\chi(p) + (v', \mathrm{d})) \oplus q' = \chi(p) \oplus (q' + (t(p), \mathrm{d}))$. By Proposition 4, $D$ has an *ng*-feasible route $r'' = \chi(p) \oplus q''$ such that $\bar{c}(r'') \leq \bar{c}(r')$ and $q'' \in \mathcal{B}$. Certainly, $cb(p) \leq cb(\chi(p)) \leq \bar{c}(r'') \leq \bar{c}(r') \leq \bar{c}(r)$.

□

To prove Proposition 2, we need to adapt Proposition 3 to work with an upper bound *ub*. This is the purpose of the next proposition.

**Proposition 2.** *For $1 \leq m \leq n+2$, let $\mathcal{F}_m$ be the family of partial routes of length $m$ obtained when the labeling algorithm with upper bound ub is executed on $D$. If $r = p \oplus (v, x) \oplus q$ is an ng-feasible route of $D$ with $\bar{c}(r) < ub$ and $|p| = m$, then $D$ has an ng-feasible route $r' = p' \oplus (v, x) \oplus q$, that satisfies conditions 1–4 of Proposition 3, such that $p' \in \mathcal{F}_m$ and $\bar{c}(r') \leq \bar{c}(r)$.*

*Proof.* Let $\mathcal{P}_m$ be the family of partial routes of length $m$ obtained when the labeling algorithm in executed on $D$. By Proposition 3, $D$ has an *ng*-feasible route $r' = p' \oplus (v, x) \oplus q$ that satisfies conditions 1–4, such that $\bar{c}(r') \leq \bar{c}(r)$ and $p' \in \mathcal{P}_m$. Certainly, $r'$ is of the form $p'' \oplus (v'', x'') \oplus q''$ for every prefix $p''$ of $p'$ (Observation 1). Then, by Proposition 1, $cb(p'') \leq \bar{c}(r') \leq \bar{c}(r) < ub$, thus $p''$ is not discarded by the labeling algorithm with upper bound *ub* and, consequently, $p' \in \mathcal{F}_m$. □

**Proposition 3.** *For $1 \leq m \leq n+2$, let $\mathcal{F}_m$ be the family of partial routes of length $m$ obtained when the labeling algorithm with upper bound ub is executed on $D$. If $r = (p + (v, x_v)) \oplus (w, x_w) \oplus q$ is an elementary route with $\bar{c}(r) < ub$, then there exists $p' \in \mathcal{F}_{|p|}$ with $t(p') = t(p)$ such that $r' = (p' + (v, x_v)) \oplus (w, x_w) \oplus q$ is ng-feasible and $cb(p' + (v, x_v)) < ub$.*

*Proof.* We consider three cases for the sake of exposition; observe that $t(p') = t(p)$ in every case.

**Case 1:** $x_v = \mathrm{d}$ and the drone is inside the truck in $p$. Then, $r = p \oplus (q + (t(p), \mathrm{d}))$ and, by Proposition 2, $D$ has an *ng*-feasible route $r' = p' \oplus (q + (t(p), \mathrm{d}))$ such that $\bar{c}(p') \leq \bar{c}(p)$ and $p' \in \mathcal{F}_{|p|}$. Moreover, as $r' = (p + (v, \mathrm{d})) \oplus q$, Proposition 1 implies that $cb(p + (v, \mathrm{d})) < ub$ as desired.

**Case 2:** $x_v = \mathrm{d}$ and the drone is outside the truck in $p$. Then, $r = p \oplus (v, \mathrm{d}) \oplus q$ and, by Proposition 2, $D$ has an *ng*-feasible route $r' = p' \oplus (v, \mathrm{d}) \oplus q$ such that $\bar{c}(r') \leq \bar{c}(r)$ and $p' \in \mathcal{F}_{|p|}$. Moreover, as $r' = (p' + (v, \mathrm{d})) \oplus q$, Proposition 1 implies that $cb(p' + (v, \mathrm{d})) \leq \bar{c}(r') \leq \bar{c}(r) < ub$ as desired.

**Case 3:** $x_v = \mathrm{t}$, thus $r = p \oplus (w, \mathrm{d}) \oplus (q + (t(p), \mathrm{t}))$. Then, by Proposition 2, $D$ has an *ng*-feasible route $r' = p' \oplus (w, \mathrm{d}) \oplus (q + (t(p), \mathrm{t}))$ with $\bar{c}(p') \leq \bar{c}(p)$ such that $p' \in \mathcal{F}_{|p|}$. Moreover, as $r' = (p' + (v, \mathrm{t})) \oplus (w, \mathrm{d}) \oplus q$, Proposition 1 implies that $cb(p' + (v, \mathrm{t})) \leq \bar{c}(r') \leq \bar{c}(r) < ub$ as desired.

□