Coupling Feasibility Pump and Large Neighborhood Search to solve the Steiner team orienteering problem

Lucas Assunção\textsuperscript{a,*}, Geraldo Robson Mateus\textsuperscript{b}

\textsuperscript{a} Departamento de Engenharia de Produção, Universidade Federal de Minas Gerais
Avenida Antônio Carlos 6627, CEP 31270-901, Belo Horizonte, MG, Brazil

\textsuperscript{b} Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
Avenida Antônio Carlos 6627, CEP 31270-901, Belo Horizonte, MG, Brazil

Abstract

The Steiner Team Orienteering Problem (STOP) is defined on a digraph in which arcs are associated with traverse times, and whose vertices are labeled as either \textit{mandatory} or \textit{profitable}, being the latter provided with rewards (profits). Given a homogeneous fleet of vehicles \( M \), the goal is to find up to \( m = |M| \) disjoint routes (from an origin vertex to a destination one) that maximize the total sum of rewards collected while satisfying a given limit on the route’s duration. Naturally, all mandatory vertices must be visited. In this work, we show that solely finding a feasible solution for STOP is NP-hard and propose a Large Neighborhood Search (LNS) heuristic for the problem. The algorithm is provided with initial solutions obtained by means of the matheuristic framework known as Feasibility Pump (FP). In our implementation, FP uses as backbone a commodity-based formulation reinforced by three classes of valid inequalities. To our knowledge, two of them are also introduced in this work. The LNS heuristic itself combines classical local searches from the literature of routing problems with a long-term memory component based on Path Relinking. We use the primal bounds provided by a state-of-the-art cutting-plane algorithm from the literature to evaluate the quality of the solutions obtained by the heuristic. Computational experiments show the efficiency and effectiveness of the proposed heuristic in solving a benchmark of 387 instances. Overall, the heuristic solutions imply an average percentage gap of only 0.54% when compared to the bounds of the cutting-plane baseline. In particular, the heuristic reaches the best previously known bounds on 382 of the 387 instances. Additionally, in 21 of these cases, our heuristic is even able to improve over the best known bounds.

\textbf{Keywords:} Orienteering problems, Feasibility Pump, Large Neighborhood Search, Path Relinking, Matheuristics, Metaheuristics

1. Introduction

The Steiner Team Orienteering Problem (STOP) arises as a more general case of the widely studied profit collection routing problem known as the Team Orienteering Problem (TOP). TOP is an NP-hard problem usually defined on a complete and undirected graph, where a value of reward (profit) is associated with each vertex, and a traverse time is associated with each edge (or arc). Given a homogeneous fleet of vehicles \( M \), TOP aims at finding up to \( m = |M| \) disjoint routes from an origin vertex to a destination one, such that each route has to satisfy a time constraint while

\textsuperscript{*}Corresponding author.

\textit{Email addresses:} lucas-assuncao@ufmg.br (Lucas Assunção), mateus@dcc.ufmg.br (Geraldo Robson Mateus)
the total sum of rewards collected is maximized. In the case of STOP, the input graph is directed, and a subset of mandatory vertices is also provided. Accordingly, STOP also aims at maximizing the total sum of rewards collected within the time limit, but, now, every mandatory vertex has to be visited.

STOP finds application, for instance, in devising the itinerary of the delivery of goods performed by shipping companies (Assunção and Mateus, 2019). Here, a reward value — which may rely on factors such as the urgency of the request and the customer priority — is associated with visiting each customer. Deliveries with top priority (e.g., those whose deadlines are expiring) necessarily have to be included in the planning. Accordingly, the goal is to select a subset of deliveries (including the top priority ones) that maximizes the total sum of rewards collected and that can be performed within a pre-established working horizon of time.

Another application arises in the planning of home health care visits. Home health care covers a wide range of services that can be given in one’s home for an illness or injury, such as monitoring a patient’s medication regimen and vital signs (blood pressure, temperature, heart rate, etc). In this case, priority could be linked to the patients’ needs. Similar applications arise in the planning of other sorts of technical visits (Tang and Miller-Hooks, 2005; Assunção and Mateus, 2019).

STOP was only introduced quite recently by Assunção and Mateus (2019). In the work, a state-of-the-art branch-and-cut algorithm from the literature of TOP is adapted to STOP, and a cutting-plane algorithm is proposed. The new algorithm relies on a compact (with a polynomial number of variables and constraints) commodity-based formulation — also introduced in the work — reinforced by some classes of inequalities, which consist of general connectivity constraints, classical lifted cover inequalities based on dual bounds and a class of conflict cuts. The cutting-plane proposed not only outperforms the baseline when solving a benchmark of STOP instances, but also solves to optimality more TOP instances than any previous algorithm in the literature.

Several variations of orienteering problems have already been addressed in the literature, combining some specific constraints, such as mandatory visits (like in STOP), multiple vehicles and time windows (Tricoire et al., 2010; Salazar-Aguilar et al., 2014; Archetti et al., 2014). Here, we discuss some of the orienteering problems that are more closely related to STOP. For instance, the single-vehicle version of STOP, known as the Steiner Orienteering Problem (SOP), was already introduced by Letchford et al. (2013). In the work, four Integer Linear Programming (ILP) models are proposed, but no computational experiment is reported.

STOP is particularly similar to the Team Orienteering Arc Routing Problem (TOARP), in which profitable and mandatory arcs are considered (instead of vertices). Archetti et al. (2014) introduced TOARP, along with a polyhedral study and a branch-and-cut algorithm able to solve instances with up to 100 vertices, 800 arcs and four vehicles. These instances consider sparse random and grid-based graphs. More recently, Riera-Ledesma and Salazar-González (2017) proposed a column generation approach that works by first converting the original customer-on-arc representation into a customer-on-vertex one, thus creating STOP-like instances. Their results indicate that the column generation algorithm is a complementary approach to the branch-and-cut of Archetti et al. (2014), being useful in those cases where the latter shows poor performance.

To the best of our knowledge, only two heuristics have been proposed for TOARP: the matheuristic of Archetti et al. (2013) and the iterated local search heuristic of Ke and Yang (2017). With respect to the single-vehicle version of TOARP, known as the Orienteering Arc Routing Problem (OARP), Archetti et al. (2016) developed a branch-and-cut algorithm based on families of facet-inducing inequalities also introduced in the work.

The specific case of STOP with no mandatory vertices — namely TOP — has drawn significant attention in the past decades, and several exact and heuristic algorithms have been proposed to solve the problem, as summarized next.
The previous exact algorithms for TOP are based on cutting planes and column generation. In summary, they consist of branch-and-cut (Dang et al., 2013a; Bianchessi et al., 2018; Hanafi et al., 2020), branch-and-price (Boussier et al., 2007; Keshtkaran et al., 2016), branch-and-cut-and-price (Poggi et al., 2010; Keshtkaran et al., 2016) and cutting-plane algorithms (El-Hajj et al., 2016; Assunção and Mateus, 2019). Among them, we highlight the branch-and-cut of Hanafi et al. (2020) as the current state-of-the-art for TOP, being able to solve to optimality five more instances than the previous state-of-the-art (Assunção and Mateus, 2019).

Regarding heuristics for TOP, several ones have been proposed for TOP, adapting a wide range of metaheuristic frameworks, such as Tabu Search (Tang and Miller-Hooks, 2005), Variable Neighborhood Search (VNS) (Archetti et al., 2007; Vansteenwegen et al., 2009), Ant Colony (Ke et al., 2008), Greedy Randomized Adaptive Search Procedure (GRASP) (Souffriau et al., 2010), Simulated Annealing (Lin, 2013), Large Neighborhood Search (LNS) (Kim et al., 2013; Vidal et al., 2015) and Particle Swarm Optimization (PSO) (Dang et al., 2013b). To our knowledge, the latest heuristic for TOP was proposed by Ke et al. (2016). Their heuristic, namely Pareto mimic algorithm, introduces a so-called mimic operator to generate new solutions by imitating incumbent ones. The algorithm also adopts the concept of Pareto dominance to update the population of incumbent solutions by considering multiple indicators that measure the quality of each solution. We refer to Vansteenwegen et al. (2011) and Gunawan et al. (2016) for detailed surveys on exact and heuristic approaches for TOP and some of its variants.

Although the heuristics proposed for TOP follow different metaheuristic frameworks, most of them have two key elements in common: the way they generate initial solutions and the local search operators used to improve the quality of the solutions, which usually consist of the classical 2-opt and its variations (Lin, 1965), coupled with vertex shiftings and exchanges between routes. While these local searches naturally apply to STOP, finding a feasible solution for STOP is not always an easy task. In TOP, feasible solutions are usually built by iteratively adding vertices to initially empty routes according to greedy criteria (Chao et al., 1996; Archetti et al., 2007; Vansteenwegen et al., 2009; Souffriau et al., 2010; Lin, 2013; Kim et al., 2013). On the other hand, in STOP, a trivial solution with empty routes is not necessarily feasible, since mandatory vertices have to be visited. In fact, as we formally prove in the remainder of this work, solely finding a feasible solution for an STOP instance (or proving its infeasibility) is NP-hard in the general case.

In this work, we propose an LNS heuristic for STOP. As far as we are aware, this is also the first heuristic devised to solve the general case of STOP. The LNS heuristic combines some of the aforementioned classical local searches from the literature of TOP and other routing problems with a long-term memory component based on Path Relinking (PR) (Glover, 1997). The proposed heuristic tackles the issue of finding an initial solution by applying the Feasibility Pump (FP) matheuristic framework of Fischetti et al. (2005). In particular, FP has been successfully used to find initial feasible solutions for several challenging problems modelled by means of Mixed Integer Linear Programming (MILP) (Berthold et al., 2019).

Given an MILP formulation for the problem addressed, FP works by sequentially and wisely rounding fractional solutions obtained from solving auxiliary problems based on the linear relaxation of the original MILP formulation. In this work, we take advantage from the compactness of the formulation presented by Assunção and Mateus (2019) to use it within the FP framework. In our experiments, we also consider a reinforced version of this formulation obtained from the addition of three classes of valid inequalities. The idea is that a stronger model, being closer to the convex hull, might help FP converging to an integer solution within less iterations. In fact, as later endorsed by our experiments, the strengthening of the original formulation pays off.

The three classes of valid inequalities adopted in this work consist of clique conflict cuts, arc-vertex inference cuts and classical lifted cover inequalities based on dual bounds. The latter one
was already used in the cutting-plane of Assunção and Mateus (2019), but, as far as we are aware, the first two classes are new valid inequalities for the problem. In particular, the clique conflict cuts generalize and extend the general connectivity constraints and cuts based on vertex conflicts used in previous works (El-Hajj et al., 2016; Bianchessi et al., 2018; Assunção and Mateus, 2019). The due proofs regarding the validity of these new inequalities and the dominance of clique conflict cuts over previously adopted inequalities are also given.

In summary, the contributions of this work are threefold: (i) the problem of finding a feasible solution for STOP is proven to be NP-hard, (ii) a previously introduced formulation for STOP is further reinforced by the addition of three classes of valid inequalities. As far as we are aware, two of them — namely clique conflict cuts and arc-vertex inference cuts — are also introduced in this work. Moreover, (iii) an LNS heuristic that uses FP to find initial solutions is presented and experimentally evaluated. To this end, we consider the primal bounds provided by the cutting-plane algorithm of Assunção and Mateus (2019) as a baseline. Our computational results show the efficiency and effectiveness of the proposed heuristic in solving a benchmark of STOP instances adapted from the literature of TOP. Overall, the heuristic solutions imply an average percentage gap of only 0.54% when compared to the bounds of the cutting-plane baseline. In particular, the heuristic reaches the best previously known bounds on 382 out of the 387 instances considered. Additionally, in 21 of these cases, the heuristic is even able to improve over the best known bounds.

The remainder of this work is organized as follows. STOP is formally defined in Section 2 along with the notation adopted throughout the work. In Section 3, we prove that solely finding a feasible solution for STOP is NP-hard. In Section 4, we describe an MILP formulation for STOP, as well as three families of inequalities used to reinforce it. In the same section, we give the proofs of validity for the inequalities (when the case) and describe the procedures used to separate them. In Section 5, we present the general cutting-plane procedure used to reinforce the original STOP formulation by the addition of the three classes of inequalities discussed in this work. Sections 6 and 7 are devoted to detailing the FP procedure used to generate initial solutions and the LNS heuristic proposed, respectively. Some implementation details are given in Section 8 followed by computational results (Section 9). Concluding remarks are provided in the last section.

2. Problem definition and notation

STOP is defined on a (not necessarily complete) digraph \( G = (N, A) \), where \( N \) is the vertex set, and \( A \) is the arc set. Let \( s, t \in N \) be the origin and the destination vertices, respectively, with \( s \neq t \). Moreover, let \( S \subseteq N\setminus\{s, t\} \) be the subset of mandatory vertices, and \( P \subseteq N\setminus\{s, t\} \) be the set of profitable vertices, such that \( N = S \cup P \cup \{s, t\} \) and \( S \cap P = \emptyset \). A reward \( p_i \in \mathbb{Z}^+ \) is associated with each vertex \( i \in P \), and a traverse time \( d_{ij} \in \mathbb{R}^+ \) is associated with each arc \( (i, j) \in A \). Additionally, a fleet \( M \) of homogeneous vehicles is available to visit the vertices, and each vehicle can run for no more than a time limit \( T \). For short, an STOP instance is defined as \( \langle G, S, P, s, t, p, d, M, T \rangle \).

The goal is to find up to \( m = |M| \) routes from \( s \) to \( t \) — one for each vehicle used — such that every mandatory vertex in \( S \) belongs to exactly one route and the total sum of rewards collected by visiting profitable vertices is maximized. Each profitable vertex in \( P \) can be visited by at most one vehicle, thus avoiding the multiple collection of a same reward.

Given a subset \( V \subseteq N \), we define the sets of arcs leaving and entering \( V \) as \( \delta^+(V) = \{(i, j) \in A : i \in V, j \in N\setminus V\} \) and \( \delta^-(V) = \{(i, j) \in A : i \in N\setminus V, j \in V\} \), respectively. Similarly, given a vertex \( i \in N \), we define the sets of vertices \( \delta^+(i) = \{j \in N : (i, j) \in A\} \) and \( \delta^-(i) = \{j \in N : (j, i) \in A\} \). Moreover, given two arbitrary vertices \( i, j \in N \) and a path \( p \) from \( i \) to \( j \) in \( G \), we define \( A_p \subseteq A \) as the arc set of \( p \).
3. Building a feasible solution for STOP is NP-hard

In TOP, a trivial feasible solution consists of a set of empty routes, one for each vehicle. This solution can be easily improved by inserting vertices while not exceeding the routes’ duration limit. In fact, the heuristics in the literature of TOP make use of this simple procedure, usually adopting greedy criteria to iteratively add vertices to the empty routes (Chao et al., 1996; Archetti et al., 2007; Vansteenwegen et al., 2009; Souffriau et al., 2010; Lin, 2013; Kim et al., 2013). Once we consider mandatory vertices (as in STOP), a trivial solution with empty routes is no longer feasible. Formally, consider the optimization problem of finding (building) a feasible solution for an arbitrary STOP instance, namely Feasibility STOP (FSTOP), defined as follows.

**(FSTOP)** Input: An STOP instance \( \langle G, S, P, s, t, p, d, M, T \rangle \).

Output: A feasible solution (if any) for the input STOP instance.

Notice that FSTOP is not interested in deciding if a given STOP instance is feasible, but in actually determining a feasibility certificate (i.e., an actual solution) when the case.

In this section, we show that FSTOP is NP-hard by proving that its decision version, namely Decision FSTOP (D-FSTOP), is already NP-hard. To this end, we reduce the Hamiltonian Path Problem (HPP) — which is known to be NP-complete (Garey and Johnson, 1979) — to FSTOP.

**(D-FSTOP)** Input: An STOP instance \( \langle G, S, P, s, t, p, d, M, T \rangle \).

Question: Is there a feasible solution for the input STOP instance?

**(HPP)** Input: A digraph \( G = (N, A) \), where \( N \) is the vertex set, and \( A \) is the arc set. An origin vertex \( s \in N \) and a destination vertex \( t \in N \). For short, \( (G, s, t) \).

Question: Is there a Hamiltonian path from \( s \) to \( t \) in \( G \)?

**Theorem 1.** \( HPP \leq_p D-FSTOP \), i.e., HPP is polynomial-time reducible to D-FSTOP.

**Proof.** First, we propose a straightforward polynomial time reduction of HPP instances to D-FSTOP ones. Given an HPP instance \( (G, s, t) \), we build a corresponding D-FSTOP instance (which is also an STOP instance) by considering the same digraph \( G \), where \( s \) and \( t \) are its origin and destination vertices as well. A single vehicle is considered, and all the vertices of \( G \) are set as mandatory (except for \( s \) and \( t \)). Accordingly, the set of profitable vertices is empty, and the traverse time vector is set to \( d = 1 \). The time limit \( T \) is a sufficiently large number (say, \( T \geq |N| - 1 \)), as to allow the selection of all mandatory vertices in a route.

Now, we only have to show that an HPP instance has an Hamiltonian path from \( s \) to \( t \) if, and only if, the corresponding D-FSTOP instance (built as described above) gives a “yes” answer. The validity of this proposition is quite intuitive, as (i) the pair of HPP and D-FSTOP instances shares the same graph, (ii) the time limit \( T \) allows that all vertices belong to an STOP solution, and (iii) by definition, the single STOP route must visit each vertex exactly once, which defines an Hamiltonian path.

**Corollary 1.** D-FSTOP and, thus, FSTOP, are NP-hard.

As a consequence of Corollary 1, heuristics for STOP are subject to an additional challenge, which consists in building an initial feasible solution in reasonable time. To address this issue, we make use of the FP matheuristic of Fischetti et al. (2005), which is described in Section 6. As a matheuristic, FP uses a mathematical formulation to guide its search for a feasible solution. Then, before describing FP, we provide an MILP formulation for STOP and discuss some families of inequalities used to further strengthen it.
4. Mathematical foundation

In the MILP formulation proposed by Assunção and Mateus (2019) for STOP — hereafter denoted by $\mathcal{F}$ — time is treated as a commodity to be spent by the vehicles when traversing each arc in their routes, such that every vehicle departs from $s$ with an initial amount of $T$ units of commodity, the time limit.

Before defining $\mathcal{F}$, let $R_{ij}$ denote the minimum time needed to reach a vertex $j$ when departing from a vertex $i$ in the graph $G$ of a given STOP instance, i.e., $R_{ij} = \min_{p} \{ \sum_{a \in A_p} d_a : p$ is a path from $i$ to $j$ in $G \}$. Accordingly, $R_{ii} = 0$ for all $i \in N$, and, if no path exists from a vertex to another, the corresponding entry of $R$ is set to infinity. One may observe that $R$ is not necessarily symmetric, since $G$ is directed. Moreover, considering that the traverse times associated with the arcs of $G$ are non-negative (and, thus, no negative cycle exists), this $R$ matrix can be computed \textit{a priori} (for each instance) by means of the classical dynamic programming algorithm of Floyd-Warshall (Cormen et al., 2001), for instance.

We introduce the following sets of variables. The binary variables $y$ and $x$ identify the solution routes, such that $y_i = 1$ if the vertex $i \in N$ is visited by a vehicle of the fleet ($y_i = 0$, otherwise), and $x_{ij} = 1$ if the arc $(i, j) \in A$ is traversed in the solution ($x_{ij} = 0$, otherwise). In addition, the continuous flow variables $f_{ij}$, for all $(i, j) \in A$, represent the amount of time still available for a vehicle after traversing the arc $(i, j)$ as not to exceed $T$. The slack variable $\varphi$ represents the number of vehicles that are not used in the solution. $\mathcal{F}$ is defined from (1) to (14).

\begin{align*}
(\mathcal{F}) & \quad \text{max} \quad \sum_{i \in P} p_i y_i, \quad (1) \\
\text{s.t.} \quad & y_i = 1 \quad \forall i \in S \cup \{s, t\}, \quad (2) \\
& \sum_{j \in \delta^+(i)} x_{ij} = y_i \quad \forall i \in S \cup P, \quad (3) \\
& \sum_{j \in \delta^+(s)} x_{sj} = \sum_{i \in \delta^-(t)} x_{is} = m - \varphi, \quad (4) \\
& \sum_{j \in \delta^+(s)} x_{is} = \sum_{j \in \delta^+(t)} x_{tj} = 0, \quad (5) \\
& \sum_{j \in \delta^+(i)} x_{ij} - \sum_{j \in \delta^-(i)} x_{ji} = 0 \quad \forall i \in S \cup P, \quad (6) \\
& f_{sj} = (T - d_{sj}) x_{sj} \quad \forall j \in \delta^+(s), \quad (7) \\
& \sum_{j \in \delta^-(i)} f_{ji} - \sum_{j \in \delta^+(i)} f_{ij} = \sum_{j \in \delta^+(i)} d_{ij} x_{ij} \quad \forall i \in S \cup P, \quad (8) \\
& f_{ij} \leq (T - R_{si} - d_{ij}) x_{ij} \quad \forall (i, j) \in A, \ i \neq s, \quad (9) \\
& f_{ij} \geq R_{ij} x_{ij} \quad \forall (i, j) \in A, \quad (10) \\
& x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A, \quad (11) \\
& y_i \in \{0, 1\} \quad \forall i \in N, \quad (12) \\
& f_{ij} \geq 0 \quad \forall (i, j) \in A, \quad (13) \\
& 0 \leq \varphi \leq m. \quad (14)
\end{align*}

The objective function in (1) gives the total reward collected by visiting profitable vertices. Constraints (2) impose that all mandatory vertices (as well as $s$ and $t$) are selected, while constraints
ensure that each vertex in \( S \cup P \) is visited at most once. Restrictions (1) ensure that at most \( m \) vehicles leave the origin \( s \) and arrive at the destination \( t \), whereas constraints (5) impose that vehicles cannot arrive at \( s \) nor leave \( t \). Moreover, constraints (6), along with constraints (2) and (3), guarantee that, if a vehicle visits a vertex \( i \in S \cup P \), then it must enter and leave this vertex exactly once. Constraints (7)-(9) ensure that each of the solution routes has a total traverse time of at most \( T \). Precisely, restrictions (7) implicitly state, along with (4), that the total flow available at the origin \( s \) is \((m - \varphi)T\), and, in particular, each vehicle (used) has an initial amount of \( T \) units of flow. Constraints (8) manage the flow consumption incurred from traversing the arcs selected, whereas constraints (9) impose that an arc \((i, j) \in A\) can only be traversed if the minimum time of a route from \( s \) to \( j \) through \((i, j)\) does not exceed \( T \). In (9), we do not consider the arcs leaving the origin, as they are already addressed by (7). Restrictions (10) are, in fact, valid inequalities that give lower bounds on the flow passing through each arc, and constraints (11)-(14) define the domain of the variables. Here, the management of the flow associated with the variables \( f \) also avoids the existence of sub-tours in the solutions.

Although the \( y \) variables can be easily discarded — as they solely aggregate specific subsets of the \( x \) variables — they enable us to represent some families of valid inequalities (as detailed in Section 4.1) by means of less dense cuts. For this reason, they are preserved. Also notice that the dimension of formulation \( F \) (both in terms of variables and constraints) is \( O(|A| + |N|) \), being completely independent from the number of vehicles available.

4.1. Families of valid inequalities

In this section, we discuss three families of valid inequalities able to reinforce \( F \). They consist of clique conflict cuts, a class of the so-called arc-vertex inference cuts and classical lifted cover inequalities based on dual bounds. To the best of our knowledge, the first and second classes of inequalities are also introduced in this work. After the description of each class of inequality, we detail their corresponding separation procedures, which are called within the cutting-plane algorithm later discussed in Section 5. Hereafter, the linear relaxation of \( F \) is denoted by \( L \).

4.1.1. Clique Conflict Cuts (CCCs)

Consider the set \( K \) of vertex pairs which cannot be simultaneously in a same valid route. Precisely, for every pair \( \{i, j\} \in K \), with \( i, j \in N \setminus \{s, t\} \), we have that any route from \( s \) to \( t \) that visits \( i \) and \( j \) (in any order) has a total traverse time that exceeds the limit \( T \). Accordingly, consider the undirected conflict graph \( G_c = (N \setminus \{s, t\}, K) \).

![Figure 1: An example of an STOP instance. Profit values are omitted.](image)

Consider the STOP instance defined by the digraph shown in Figure 1, whose arcs have traverse times \( d_{si} = d_{sl} = d_{ik} = d_{ki} = d_{lt} = d_{jt} = 1 \) and \( d_{kj} = d_{lj} = 2 \). In this case, \( S = \emptyset \) (i.e., there
is no mandatory vertex), and a single vehicle is available to move from $s$ to $t$, with $T = 4$. The conflict graph related to this instance is given in Figure 2.

![Conflict graph related to the STOP instance of Figure 1.](image)

Now, let $\Sigma$ be the set of all the cliques of $G_c$, which are referred to as conflict cliques. Then, CCCs are defined as

$$
\sum_{e \in \delta^-(V)} x_e \geq \sum_{i \in \sigma} y_i \quad \forall \sigma \in \Sigma, \forall V \subseteq N \setminus \{s\}, \sigma \subseteq V, \tag{15}
$$

$$
\sum_{e \in \delta^+(V)} x_e \geq \sum_{i \in \sigma} y_i \quad \forall \sigma \in \Sigma, \forall V \subseteq N \setminus \{t\}, \sigma \subseteq V. \tag{16}
$$

Intuitively speaking, CCCs state that the number of visited vertices from any conflict clique gives a lower bound on the number of routes needed to make a solution feasible for STOP. In other words, vertices of a conflict clique must necessarily belong to different routes in any feasible solution for $F$. The formal proof of the validity of CCCs is given as follows.

**Proposition 1.** Inequalities (15) do not cut off any feasible solution of $F$.

**Proof.** Consider an arbitrary feasible solution $(\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})$ for $F$, a conflict clique $\sigma \in \Sigma$ and a subset $V \subseteq N \setminus \{s\}$, with $\sigma \subseteq V$. Then, we have two possibilities:

1. if $\sum_{i \in \sigma} \bar{y}_i = 0$, then $\sum_{e \in \delta^-(V)} x_e \geq \sum_{i \in \sigma} \bar{y}_i$.

2. if $\sum_{i \in \sigma} \bar{y}_i > 0$, then, since $s \notin V$, there must be a subset of arcs $A' \subseteq \delta^-(V)$, with $\bar{x}_e = 1$ for all $e \in A'$, and $|A'| = \sum_{i \in \sigma} \bar{y}_i$, so that every vertex $i \in \sigma$ having $\bar{y}_i = 1$ is traversed in a different route from $s$. Then, also in this case, $\sum_{e \in \delta^-(V)} x_e \geq \sum_{i \in \sigma} \bar{y}_i$. \hfill \square

**Corollary 2.** Inequalities (16) do not cut off any feasible solution of $F$.

**Proof.** The same mathematical argumentation of Proposition 1 can be used to prove the validity of inequalities (16) by simply replacing $s$ and $\delta^-(V)$ with $t$ and $\delta^+(V)$, respectively. \hfill \square

Notice that, from (15), we have that $\sum_{e \in \delta^-(V)} x_e = \sum_{e \in \delta^+(V)} x_e$ for all $V \subseteq S \cup P = N \setminus \{s, t\}$. Therefore, for all $V \subseteq N \setminus \{s, t\}$, inequalities (15) and (16) cut off the exact same regions of the
polyhedron \( \mathcal{L} \), the linearly relaxed version of \( \mathcal{F} \). In this sense, the whole set of CCCs can be represented in a more compact manner by (15) and

\[
\sum_{e \in \delta^+(V)} x_e \geq \sum_{i \in \sigma} y_i \quad \forall \sigma \in \Sigma, \forall V \subseteq N \setminus \{t\}, \sigma \cup \{s\} \subseteq V.
\]  

(17)

**Theorem 2.** For every non-maximal conflict clique \( \sigma_1 \in \Sigma \), there is a clique \( \sigma_2 \in \Sigma \), with \( \sigma_1 \subset \sigma_2 \), such that the CCCs referring to \( \sigma_1 \) are dominated by the ones referring to \( \sigma_2 \), when considering the polyhedron \( \mathcal{L} \).

**Proof.** Consider an arbitrary non-maximal conflict clique \( \sigma_1 \in \Sigma \) and, without loss of generality, a solution \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\) for \( \mathcal{L} \) that is violated by at least one of the CCCs referring to \( \sigma_1 \). We have to show that there exists a clique \( \sigma_2 \in \Sigma \), with \( \sigma_1 \subset \sigma_2 \), whose corresponding CCCs also cut off \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\) from \( \mathcal{L} \). To this end, consider a vertex \( k \in (S \cup P)\setminus \sigma_1 \), such that \( \{k, j\} \in \mathcal{K} \) for all \( j \in \sigma_1 \). Such vertex exists, since \( \sigma_1 \) is supposed to be non-maximal. Then, define the conflict clique \( \sigma_2 = \sigma_1 \cup \{k\} \).

By assumption, \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\) violates at least one of the CCCs (15) and (16) referring to \( \sigma_1 \). Then, we must have at least one of these two possibilities:

1. \( \exists V' \subseteq N \setminus \{s\} \), with \( \sigma_1 \subseteq V' \), such that

\[
\sum_{e \in \delta^-(V')} \bar{x}_e < \sum_{i \in \sigma_1} \bar{y}_i.
\]

Define the set \( V'' = V' \cup \{k\} \). Notice that, if we originally have \( k \in V' \), then the CCC (15) considering \( \sigma_2 \) and \( V = V' (= V''\) in this case) is also violated by \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\). Otherwise, if \( k \notin V' \), then

\[
\sum_{e \in \delta^-(V'')} \bar{x}_e = \sum_{e \in \delta^-(V')} \bar{x}_e + \sum_{i \in \delta^-(k) \setminus V'} \bar{x}_{ik} - \sum_{j \in \delta^+(k) \cap V'} \bar{x}_{kj},
\]

(18)

which follows from the fact that the difference between the summation of arcs entering \( V'' \) and that of arcs entering \( V' \) corresponds to the difference between (g) the sum of arcs arriving at \( k \) that do not leave a vertex of \( V' \) and (h) the sum of arcs leaving \( k \) that arrive at a vertex of \( V' \). In other words, (g) considers the arcs that traverse the cut \((N \setminus V'', V'') \) but do not traverse \((N \setminus V', V') \), while (h) considers the arcs that do not traverse the cut \((N \setminus V'', V'') \) but traverse \((N \setminus V', V') \).

From (18) and the hypothesis that \( \sum_{e \in \delta^-(V')} \bar{x}_e < \sum_{i \in \sigma_1} \bar{y}_i \), it follows that

\[
\sum_{e \in \delta^-(V'')} \bar{x}_e < \sum_{i \in \sigma_1} \bar{y}_i + \sum_{i \in \delta^-(k) \setminus V'} \bar{x}_{ik} - \sum_{j \in \delta^+(k) \cap V'} \bar{x}_{kj}.
\]

(19)

From (3), (6), (11) and (12), we have that

\[
\sum_{i \in \delta^-(k) \setminus V'} \bar{x}_{ik} - \sum_{j \in \delta^+(k) \cap V'} \bar{x}_{kj} \leq \bar{y}_k.
\]

(20)
Then, from (19), it follows that
\[
\sum_{e \in \delta^-(V')} \bar{x}_e < \sum_{i \in \sigma_1} \bar{y}_i + \bar{y}_k < \sum_{i \in \sigma_2} \bar{y}_i, \tag{21}
\]

Since \(\sigma_2 = \sigma_1 \cup \{k\}\) and \(\sigma_1 \subseteq V'\), we have that \(\sigma_2 \subseteq V''\). Then, also in this case, there exists a CCC (15) referring to \(\sigma_2\) (in particular, with \(V = V''\)) that is also violated by \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\).

2. \(\exists V' \subseteq N \setminus \{t\}\), with \(\sigma_1 \subseteq V'\), such that \(\sum_{e \in \delta^+(V')} \bar{x}_e < \sum_{i \in \sigma_1} \bar{y}_i\).

Through the same idea of the previous case, we can also show that the CCC (16) referring to \(\sigma_2\) that considers \(V = V' \cup \{k\}\) is also violated by \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\).

From Theorem 2, we may discard several CCCs (15) and (16), as we only need to consider the ones related to maximal conflict cliques. We highlight that, in the conflict graph, there might be maximal cliques of size one, which correspond to isolated vertices. We also remark that CCCs are a natural extension of two classes of valid inequalities previously addressed in the literature of TOP and STOP, namely general connectivity constraints (Bianchessi et al., 2018) and conflict cuts (Assunção and Mateus, 2019). In particular, these two classes consist of CCCs based on cliques of sizes one and two, respectively. Then, the result below is another direct implication of Theorem 2.

**Corollary 3.** CCCs dominate the general connectivity constraints of (Bianchessi et al., 2018) and the conflict cuts of (Assunção and Mateus, 2019).

### 4.1.2. Separation of CCCs

Let \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\) be a given fractional solution referring to \(L\), the linear relaxation of \(F\). Also consider the residual graph \(\bar{G} = (N, \bar{A})\) induced by \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\), such that each arc \((i, j) \in A\) belongs to \(\bar{A}\) if, and only if, \(\bar{x}_{ij} > 0\). Moreover, a capacity \(c[i,j] = \bar{x}_{ij}\) is associated with each arc \((i, j) \in A\).

As detailed in the sequel, the separation of CCCs involves solving maximum flow problems. In this sense, consider the notation defined as follows. Given an arbitrary digraph \(G_a\) with capacitated arcs, and two vertices \(i\) and \(j\) of \(G_a\), let max-flow\(_{i \rightarrow j}\)(\(G_a\)) denote the problem of finding the maximum flow (and, thus, a minimum cut) from \(i\) to \(j\) on \(G_a\). Moreover, let \(\{F_{i \rightarrow j}, \theta_{i \rightarrow j}\}\) denote an optimal solution of such problem, where \(F_{i \rightarrow j}\) is the value of the maximum flow, and \(\theta_{i \rightarrow j}\) defines a corresponding minimum cut, with \(i \in \theta_{i \rightarrow j}\).

Considering the shortest (minimum time) paths matrix \(R\) defined in the beginning of Section 4, we first compute the set \(K\) of conflicting vertices by checking, for all pairs \(\{i, j\}, i, j \in N \setminus \{s, t\}, i \neq j\), if there exists a path from \(s\) to \(t\) on \(G\) that traverses both \(i\) and \(j\) (in any order) and that satisfies the total time limit \(T\). If no such path exists, then \(\{i, j\}\) belongs to \(K\). For simplicity, in this work, we only consider a subset \(\bar{K} \subseteq K\) of conflicting vertex pairs, such that

\[
\{i, j\} \in \bar{K}\text{ iff } \begin{cases} 
(i) & R_{si} + R_{ij} + R_{jt} > T, \quad \text{and} \\
(ii) & R_{sj} + R_{ji} + R_{it} > T, \quad \forall i, j \in N \setminus \{s, t\}, i \neq j.
\end{cases}
\]

where (i) is satisfied if a minimum traverse time route from \(s\) to \(t\) that visits \(i\) before \(j\) exceeds the time limit. Likewise, (ii) considers a minimum time route that visits \(j\) before \(i\). Since the routes
from $s$ to $t$ considered in (i) and (ii) are composed by simply aggregating entries of $R$, they may not be elementary, i.e., they might visit a same vertex more than once. Then, $\tilde{K}$ is not necessarily equal to $K$. Also observe that we only have to compute $\tilde{K}$ a single time for a given STOP instance, as it is completely based on the original graph $G$.

After that, we build the corresponding conflict graph $\tilde{G}_c = (N \setminus \{s, t\}, \tilde{K})$. Thereafter, we compute a subset $\Sigma \subseteq \Sigma$ of conflict cliques by finding all the maximal cliques of $\tilde{G}_c$, including the ones of size one. To this end, we apply the depth-first search algorithm of Tomita et al. (2006), which runs with worst-case time complexity of $O(3^{\sqrt{s}})$.

| Input: | A fractional solution $(\bar{x}, \bar{y}, \bar{f}, \varphi)$, its corresponding residual graph $\tilde{G} = (N, \tilde{A})$ and the subset $\Sigma \subseteq \Sigma$ of maximal conflict cliques. |
| Output: | A set $\mathcal{X}'$ of CCCs violated by $(\bar{x}, \bar{y}, \bar{f}, \varphi)$. |
| 1. $\mathcal{X}' \leftarrow \emptyset$; | 2. Set all cliques in $\Sigma$ as active; |
| 3. for all $(\sigma \in \Sigma)$ do | 4. if (\sigma is active) then |
| 5. Step I. Building the auxiliary graphs | 6. Build $G'_1 = (\tilde{N}_1', \tilde{A}_1')$, $\tilde{N}_1' = N' \cup \{\beta_1\}$, $\tilde{A}_1' = \tilde{A} \cup \{(i, \beta_1) : i \in \sigma\}$; |
| 7. Build $G'_2 = (\tilde{N}_2', \tilde{A}_2')$, $\tilde{N}_2' = N \cup \{\beta_2\}$, $\tilde{A}_2' = \{(v, u) \in \tilde{A} \cup \{(i, \beta_2) : i \in \sigma\}$; |
| 8. for all $(u, v) \in \tilde{A}$ do | 9. $c'_1[u, v] \leftarrow c'_2[u, v] \leftarrow c[v, u]$; |
| 10. end-for; | 11. for all $(i \in \sigma)$ do |
| 12. $c'_i[i, \beta_1] \leftarrow c'_i[i, \beta_2] \leftarrow |M|$; |
| 13. end-for; | 14. Step II. Looking for a violated CCC (15) |
| 15. \{\text{\textit{F}}_{s \rightarrow \beta_1}, \theta_{s \rightarrow \beta_1}\} \leftarrow \text{max-flow}_{s \rightarrow \beta_1}(G'_1)$; |
| 16. if $(\text{\textit{F}}_{s \rightarrow \beta_1} < \sum_{i \in \sigma} \bar{y}_i)$ then | 17. $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{\{N \setminus \theta_{s \rightarrow \beta_1}, \sigma\}\}$; |
| 18. Call update-active-cliques($\Sigma$, $\sigma$, $\bar{y}$); |
| 19. else | 20. Step III. Looking for a violated CCC (16) |
| 21. \{\text{\textit{F}}_{t \rightarrow \beta_2}, \theta_{t \rightarrow \beta_2}\} \leftarrow \text{max-flow}_{t \rightarrow \beta_2}(G'_2)$; |
| 22. if $(\text{\textit{F}}_{t \rightarrow \beta_2} < \sum_{i \in \sigma} \bar{y}_i)$ then | 23. $\mathcal{X}' \leftarrow \mathcal{X}' \cup \{\{N \setminus \theta_{t \rightarrow \beta_2}, \sigma\}\}$; |
| 24. Call update-active-cliques($\Sigma$, $\sigma$, $\bar{y}$); |
| 25. end-if; | 26. end-if-else |
| 27. end-if; | 28. end-for; |
| 29. return $\mathcal{X}'$; |

Figure 3: Algorithm used to separate violated CCCs.

Once $\bar{\Sigma}$ is computed, we look for violated CCCs of types (15) and (16) as detailed in Figure 3. Let the $\mathcal{X}'$ keep the CCCs found during the separation procedure. Initially, $\mathcal{X}'$ is empty (line 1, Figure 3). Due to the possibly large number of maximal cliques, we adopt a simple filtering mechanism to discard cliques from further search whenever convenient. Precisely, the cliques, which are initially marked as active (line 2, Figure 3), are disabled if any of its vertices belongs to
a previously separated CCC. Then, for every conflict clique \( \sigma \in \tilde{\Sigma} \), we check if it is currently active (lines 3 and 4, Figure 3) and, if so, we build two auxiliary graphs, one for each type of CCC. The first graph, denoted by \( \tilde{G}'_1 \), is built by adding to the residual graph \( \tilde{G} \) an artificial vertex \( \beta_1 \) and \(|\sigma|\) arcs \((i, \beta_1)\), one for each \( i \in \sigma \) (see line 6, Figure 3). The second one, denoted by \( \tilde{G}'_2 \), is built by reversing all the arcs of \( \tilde{G} \) and, then, adding an artificial vertex \( \beta_2 \), as well as an arc \((i, \beta_2)\) for each \( i \in \sigma \) (see line 7, Figure 3).

The capacities of the arcs of \( \tilde{G}'_1 \) and \( \tilde{G}'_2 \) are kept in the data structures \( c'_1 \) and \( c'_2 \), respectively. In both graphs, the capacities of the original arcs in \( \tilde{G} \) are preserved (see lines 8-10, Figure 3). Moreover, all the additional arcs have a same capacity value, which is equal to a sufficiently large number. Here, we adopted the value of \(|M|\), the number of vehicles (see lines 11-13, Figure 3).

Figure 4 illustrates the construction of the auxiliary graphs described above. In this example, we consider the STOP instance of Figure 1 and assume that the current fractional solution \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\) has \(\bar{x}_{si} = \bar{x}_{ik} = \bar{x}_{kj} = \bar{x}_{sl} = 0.5, \bar{x}_{ki} = \bar{x}_{il} = \bar{x}_{lt} = 0\) and \(\bar{x}_{jt} = 1\). Moreover, we consider the maximal conflict clique \(\{i, k, j\}\) (see, once again, the conflict graph of Figure 2).

![Diagram](image)

(a) Example of an auxiliary graph \( \tilde{G}'_1 \) used in the separation of CCCs (15).

(b) Example of an auxiliary graph \( \tilde{G}'_2 \) used in the separation of CCCs (16).

Figure 4: Auxiliary graphs built when considering the STOP instance of Figure 1, the maximal conflict clique \(\{i, k, j\}\) and a fractional solution \(\bar{x}\), with \(\bar{x}_{si} = \bar{x}_{ik} = \bar{x}_{kj} = \bar{x}_{sl} = \bar{x}_{lj} = 0.5, \bar{x}_{ki} = \bar{x}_{il} = \bar{x}_{lt} = 0\) and \(\bar{x}_{jt} = 1\). Here, the values associated with the arcs are their corresponding capacities, and the infinity symbol stands for a sufficiently large value.

Once the auxiliary graphs are built for a given \( \sigma \in \tilde{\Sigma} \), the algorithm first looks for a violated
CCC (15) by computing the maximum flow from $s$ to $\beta_1$ on $\tilde{G}_1'$. Accordingly, let $\{F_{s\rightarrow \beta_1}, \theta_{s\rightarrow \beta_1}\}$ be the solution of $\text{max-flow}_{s\rightarrow \beta_1}(\tilde{G}_1')$. Recall that $F_{s\rightarrow \beta_1}$ gives the value of the resulting maximum flow, and $\theta_{s\rightarrow \beta_1}$ defines a corresponding minimum cut, with $s \in \theta_{s\rightarrow \beta_1}$. Then, the algorithm checks if $F_{s\rightarrow \beta_1}$ is smaller than $\sum_{i \in \sigma} \bar{y}_i$. If that is the case, a violated CCC (15) is identified and added to $X'$. Precisely, this inequality is denoted by $\{N \setminus \theta_{s\rightarrow \beta_1}, \sigma\}$ and defined as

$$\sum_{e \in \delta^-(N \setminus \theta_{s\rightarrow \beta_1})} x_e \geq \sum_{i \in \sigma} y_i,$$  

(23)

where $N \setminus \theta_{s\rightarrow \beta_1}$ corresponds to the subset $V \subseteq N \setminus \{s\}$ of (15). If a violated CCC (15) is identified, we also disable the cliques that are no longer active by calling the procedure $\text{update-active-cliques}$, which is described in Figure 5. The separation of CCCs (15) is summarized at lines 14-19, Figure 3.

4.1.3. Arc-Vertex Inference Cuts (AVICs)

Consider the set $E = \{(i, j) : (i, j) \in A \land (j, i) \in A\}$. AVICs are defined as

$$|y_i - y_j| \leq 1 - (x_{ij} + x_{ji}) \quad \forall \{i, j\} \in E,$$

(25)

which can be linearized as

$$y_i - y_j \leq 1 - (x_{ij} + x_{ji}) \quad \forall \{i, j\} \in E, \quad \text{and}$$

(26)

$$y_j - y_i \leq 1 - (x_{ij} + x_{ji}) \quad \forall \{i, j\} \in E.$$

(27)

Figure 5: Procedure $\text{update-active-cliques}$, which manages the currently active conflict cliques during the separation of CCCs.

Notice that, if the separation algorithm does not find a violated CCC (15) for the current clique, then this clique remains active. In this case, the algorithm looks for a violated CCC of type (16) by computing $\{F_{t\rightarrow \beta_2}, \theta_{t\rightarrow \beta_2}\} = \text{max-flow}_{t\rightarrow \beta_2}(\tilde{G}_2')$. If $F_{t\rightarrow \beta_2}$ is smaller than $\sum_{i \in \sigma} \bar{y}_i$, then a violated CCC (16) is identified and added to $X'$. This CCC is denoted by $\{N \setminus \theta_{t\rightarrow \beta_2}, \sigma\}$ and defined as

$$\sum_{e \in \delta^+(N \setminus \theta_{t\rightarrow \beta_2})} x_e \geq \sum_{i \in \sigma} y_i.$$

(24)

Here, $N \setminus \theta_{t\rightarrow \beta_2}$ corresponds to the subset $V \subseteq N \setminus \{t\}$ of (16). Also in this case, we disable the cliques that are no longer active by calling the procedure $\text{update-active-cliques}$. The separation of CCCs (16) is summarized at lines 20-25, Figure 3.
In logic terms, these inequalities correspond to the boolean expressions
\[(x_{ij} = 1 \oplus x_{ji} = 1) \rightarrow y_i - y_j = 0 \quad \forall \{i, j\} \in E,\]
where \(\oplus\) stands for the exclusive disjunction operator.

The validity of AVICs relies on two trivial properties that are inherent to feasible solutions for \(F\): (i) for all \(\{i, j\} \in E\), arcs \((i, j)\) and \((j, i)\) cannot be simultaneously selected (i.e., \(x_{ij} + x_{ji} \leq 1\)), and (ii) once an arc \((i, j) \in A\) is selected in a solution (i.e., \(x_{ij} = 1\)), we must have \(y_i = y_j\) (which, more precisely, are also equal to one).

One may notice that simpler valid inequalities can be devised by considering arcs separately, as follows
\[|y_i - y_j| \leq 1 - x_{ij} \quad \forall \{i, j\} \in A. \tag{28}\]
However, these inequalities are not only weaker, but redundant for formulation \(F\), as proven next.

**Proposition 2.** Inequalities \((28)\) do not cut off any solution from the polyhedron \(L\).

**Proof.** From \((4)\) and \((6)\) and the domain of the binary variables \(x\) and \(y\), we have that \(0 \leq x_{ij} \leq y_i \leq 1\) and \(0 \leq x_{ij} \leq y_j \leq 1\) for all \((i, j) \in A\). Accordingly, the maximum possible value assumed by \(|y_i - y_j|\) occurs when one of the corresponding \(y\) variables assumes its minimum value (i.e., \(x_{ij}\)), and the other assumes its maximum (i.e., 1). \(\Box\)

4.1.4. Separation of AVICs

Notice that the total number of AVICs \((26)\) and \((27)\) is equal to \(2 \times |E|\), which is at most \(|A|\). Then, this family of inequalities can be separated by complete enumeration with \(O(|A|)\) time complexity.

4.1.5. Lifted Cover Inequalities (LCIs)

First, consider the knapsack inequality
\[\sum_{i \in P} p_i y_i \leq \lceil \tau \rceil, \tag{29}\]
where \(\tau\) is a dual (upper) bound on the optimal solution value of \(F\), and \(P \subseteq N \setminus \{s, t\}\) is the set of profitable vertices, with \(p_i \in \mathbb{Z}^+\) for all \(i \in P\), as defined in Section 2. By definition, \((29)\) is valid for \(F\), once its left-hand side corresponds to the objective function of this formulation.

Based on \((29)\), we devise LCIs, which are classical cover inequalities strengthened through lifting (see, e.g., Balas (1975); Wolsey (1975); Gu et al. (1998); Kaparis and Letchford (2008)). Precisely, we depart from cover inequalities of the form
\[\sum_{i \in C} y_i \leq |C| - 1, \tag{30}\]
where the set \(C \subseteq P\) covers \((29)\), i.e., \(\sum_{i \in C} p_i > \lceil \tau \rceil\). Moreover, \(C\) must be minimal, i.e., \(\sum_{i \in C \setminus \{j\}} p_i \leq \lceil \tau \rceil\) for all \(j \in C\). Then, let the disjoint sets \(C_1\) and \(C_2\) define a partition of \(C\), with \(C_1 \neq \emptyset\). LCIs are defined as
\[\sum_{i \in C_1} y_i + \sum_{j \in C_2} \pi_j y_j + \sum_{j \in P \setminus C} \mu_j y_j \leq |C_1| + \sum_{j \in C_2} \pi_j - 1, \tag{31}\]
where \(\pi_i \in \mathbb{Z}, \pi_i \geq 1\) for all \(i \in C_2\), and \(\mu_j \in \mathbb{Z}, \mu_i \geq 0\) for all \(i \in P \setminus C\), are the lifted coefficients. One may note that setting \(\pi = 1\) and \(\mu = 0\) reduces \((31)\) to the classical cover inequality \((30)\).
4.1.6. Separation of LCIs

The separation algorithm we adopt follows the classical algorithmic framework of Gu et al. (1998). In general terms, the liftings are done sequentially (i.e., one variable at a time) according to the values of the \( y \) variables in the current fractional solution \((\bar{x}, \bar{y}, \bar{f}, \bar{\varphi})\), and the lifted coefficients are computed by solving auxiliary knapsack problems to optimality. Once all the variables are tested for lifting, the resulting LCI is checked for violation. We refer to Assunção and Mateus (2019) for a detailed explanation of the separation algorithm and a didactic overview on the concepts of lifting.

5. Reinforcing the original formulation through cutting planes

In this section, we briefly describe the cutting-plane algorithm used to reinforce formulation \( \mathcal{F} \) through the addition of the valid inequalities discussed in Section 4.1. The algorithm follows the same framework adopted by Assunção and Mateus (2019), except for the types of cuts separated. For simplicity, we assume that \( \mathcal{F} \) is feasible.

The algorithm starts by adding to \( \mathcal{L} \) — the linear relaxation of \( \mathcal{F} \) — all AVICs, which are separated by complete enumeration (as discussed in Section 4.1.4). Then, an iterative procedure takes place. Precisely, at each iteration, the algorithm looks for CCCs and LCIs violated by the solution of the current LP model. These cuts are found by means of the separation procedures described in Sections 4.1.2 and 4.1.6. Instead of selecting all the violated CCCs found, we only add to the model the most violated cut (if any) and the ones that are sufficiently orthogonal to it. Such strategy is able to balance the strength and diversity of the cuts separated, while limiting the model size (see, e.g., Wesselmann and Suhl (2012); Samer and Urrutia (2015); Bicalho et al. (2016)). Naturally, this filtering procedure does not apply to LCIs, since at most a single LCI is separated per iteration. Details on how these cuts are selected are given in Section 8.

The algorithm iterates until either no more violated cuts are found or the bound improvement of the current model — with respect to the model from the previous iteration — is inferior or equal to a tolerance \( \epsilon_1 \). The order in which CCCs and LCIs are separated is not relevant, since the bound provided by the current LP model is only updated at the end of each loop, when all separation procedures are done.

6. Feasibility Pump (FP)

In this section, we describe the FP algorithm we adopt to find initial solutions for STOP by means of formulation \( \mathcal{F} \) and its reinforced version obtained from the cutting-plane algorithm previously presented. The FP matheuristic was proposed by Fischetti et al. (2005) as an alternative to solve the NP-hard problem of finding feasible solutions for generic MILP problems of the form

\[
\min \{ c^T x : Ax \geq b, x_i \text{ integer } \forall i \in I \}.
\]

Here, \( x \) and \( c \) are column vectors of, respectively, variables and their corresponding costs, \( A \) is the restriction matrix, \( b \) is a column vector, and \( I \) is the set of integer variables.

At each iteration, also called pumping cycle (or, simply, pump) of FP, an integer (infeasible) solution \( \tilde{x} \) is used to build an auxiliary LP problem based on the linear relaxation of the original MILP problem. Precisely, the auxiliary problem aims at finding a solution \( x^* \) with minimum distance from \( \tilde{x} \) in the search space defined by \( \{ x : Ax \geq b \} \). Each new \( x^* \) is rounded and used as the integer solution of the next iteration. The algorithm ideally stops when the current solution of the auxiliary problem is also integer (i.e., \([x^*] = x^*\), where \([x^*] = [x^* + 0.5] \)) and, thus, feasible for the original problem. Notice that FP only works in the continuous space of solutions that satisfy all the linear constraints of the original problem, and the objective function of the auxiliary problems is the one element that guides the fractional solutions into integer feasibility.
The original FP framework pays little attention to the quality of the solutions. In fact, the objective function of the original MILP problem is only taken into account to generate an initial fractional solution to be rounded and used in the first iteration. In all the subsequent iterations, the auxiliary LPs aim at minimizing distance functions that do not explore the original objective, which explains the poor quality of the solutions obtained (Fischetti et al., 2005; Achterberg and Berthold, 2007). Some variations of the framework address this issue by combining, in the auxiliary problems, the original objective function with the distance metric. That is the case, for instance, of the Objective Feasibility Pump (OFP) (Achterberg and Berthold, 2007), in which the transition from the original objective function to the distance-based auxiliary one is done gradually with the progress of the pumps. We refer to Berthold et al. (2019) for a detailed survey on the several FP variations that have been proposed throughout the years to address possible drawbacks and convergence issues of the original framework.

In this work, we adopt both the original FP framework and OFP to find feasible solutions for $\mathcal{F}$. In the sequel, we only describe in details OFP, as it naturally generalizes FP. For simplicity, formulation $\mathcal{F}$ is used throughout the explanation, instead of a generic MILP.

Consider the vector $x$ of decision variables (one for each arc in $A$), as defined in Section 4 and let $\tilde{x} \in \{0,1\}^{|A|}$ be a binary vector defining a not necessarily feasible solution for $\mathcal{F}$. The distance function used to guide the OFP framework into integer feasibility is defined as

$$\Delta(x, \tilde{x}) = \sum_{(i,j) \in A} |x_{ij} - \tilde{x}_{ij}|,$$

which can be rewritten in a linear manner as

$$\Delta(x, \tilde{x}) = \sum_{(i,j) \in A: \tilde{x}_{ij}=0} x_{ij} + \sum_{(i,j) \in A: \tilde{x}_{ij}=1} (1-x_{ij}).$$

Considering the $y$ decision variables on the selection of vertices in the solution routes (as defined in Section 4), the objective function of the auxiliary problems solved at each iteration of OFP consists of a convex combination of the distance function $\Delta(x, \tilde{x})$ and the original objective function of $\mathcal{F}$. Precisely,

$$\Delta_{\gamma}(x, y, \tilde{x}) = \frac{(1-\gamma)}{\|\Delta(x, \tilde{x})\|} \Delta(x, \tilde{x}) + \frac{\gamma}{\|\sum_{i \in P} p_i y_i\|} \min \left( \sum_{i \in P} p_i y_i \right),$$

with $\gamma \in [0, 1]$, and $\|\cdot\|$ being the Euclidean norm of a vector. Notice that, in (34), we consider an alternative definition of $\mathcal{F}$ as a minimization problem, in which $\max \sum_{i \in P} p_i y_i = \min(-\sum_{i \in P} p_i y_i)$.

Moreover, both $\Delta(x, \tilde{x})$ and the original objective function are normalized in order to avoid scaling issues. Also notice that, since $\tilde{x}$ is a constant vector, $\|\Delta(x, \tilde{x})\| = \sqrt{|A|}$ in this case.

Now, consider the polyhedron $\Omega$ defined by the feasible region of $\mathcal{L}$, the linear relaxation of $\mathcal{F}$. Precisely, $\Omega = \{(x, y, f, \varphi) \in \mathbb{R}^{|A|} \times \mathbb{R}^{|N|} \times \mathbb{R}^{|A|} : (2)-(10), (14), 0 \leq x \leq 1, 0 \leq y \leq 1 \text{ and } f \geq 0\}$. OFP works by iteratively solving auxiliary problems defined as

$$D(x, y, \tilde{x}, \gamma) : \min \{\Delta_{\gamma}(x, y, \tilde{x}) : (x, y, f, \varphi) \in \Omega\},$$

where $\gamma$ balances the influence of the distance function and the original objective, i.e., the integer feasibility and the quality of the solution. Considering $\mathcal{F}$, the OFP algorithm is described in Figure 6.
**Input:** The model $\mathcal{F}$, $\max_pumps \in \mathbb{Z}^+$, $\max_pumps \geq 1$, $\lambda \in [0, 1]$ and $K \in \mathbb{Z}^+$.

**Output:** Ideally, a feasible solution for $\mathcal{F}$.

1. Initialize $\gamma \leftarrow 1$ and $\text{iter\_counter} \leftarrow 1$;
2. Solve $D(x, y, 0, \gamma)$, obtaining a solution $(x^*, y^*, f^*, \varphi^*)$;
3. if $x^*$ is integer then return $x^*$;
4. $\tilde{x} \leftarrow [x^*]$ (= rounding of $x^*$);
5. while ($\text{iter\_counter} \leq \max_pumps$);
   6. Update $\gamma \leftarrow \lambda \gamma$ and $\text{iter\_counter} \leftarrow \text{iter\_counter} + 1$;
   7. Solve $D(x, y, \tilde{x}, \gamma)$, obtaining a solution $(x^*, y^*, f^*, \varphi^*)$;
   8. if $(x^*$ is integer) then return $x^*$;
   9. if $(\tilde{x} \neq [x^*])$ then $\tilde{x} \leftarrow [x^*]$;
10. else flip $\text{rand}(K/2, 3K/2)$ entries $\tilde{x}_{ij}$, $(i, j) \in A$, with highest $|x^*_{ij} - \tilde{x}_{ij}|$;
11. end-while;
12. return 0;

Figure 6: Description of the OFP algorithm when considering formulation $\mathcal{F}$.

Aside from the corresponding model $\mathcal{F}$, the algorithm receives as input three values: the maximum number of iterations (pumps) to be performed ($\max_pumps$), a rate by which the $\gamma$ value is decreased at each pump ($\lambda$) and a basis value ($K$) used to compute the amplitude of the perturbations to be performed in solutions that cycle.  

At the beginning, $\gamma$ and a variable that keeps the number of the current iteration ($\text{iter\_counter}$) are both set to one (line 1, Figure 6). Then, the current problem $D(x, y, 0, \gamma)$ is solved, obtaining a solution $(x^*, y^*, f^*, \varphi^*)$. Notice that, since $\gamma = 1$ at this point, $D(x, y, 0, \gamma)$ corresponds to $\mathcal{L}$, and the integer solution $0$ plays no role. If $x^*$ is integer, and, thus, the current solution is feasible for $\mathcal{F}$, the algorithm stops. Otherwise, the rounded value of $x^*$ is kept in a vector $\tilde{x}$ (see lines 2-4, Figure 6).

After the first pump, an iterative procedure takes place until either an integer feasible solution is found or the maximum number of iterations is reached. At each iteration, the $\gamma$ value is decreased by the fixed rate $\lambda$, and the iteration counter is updated. Then, $D(x, y, \tilde{x}, \gamma)$ is solved, obtaining a solution $(x^*, y^*, f^*, \varphi^*)$. If $x^*$ is integer at this point, the algorithm stops. Otherwise, it checks if the algorithm is caught up in a cycle of size one, i.e., if $\tilde{x}$ (the rounded solution from the previous iteration) is equal to $[x^*]$. If not, $\tilde{x}$ is simply updated to $[x^*]$. In turn, if a cycle is detected, the algorithm performs a perturbation on $\tilde{x}$. Precisely, a random integer in the open interval $(K/2, 3K/2)$ is selected as the quantity of binary entries $\tilde{x}_{ij}$, $(i, j) \in A$, to be flipped to the opposite bound. This perturbation prioritizes entries that have highest values in the distance vector $|x^* - \tilde{x}|$. The loop described above is summarized at lines 5-11, Figure 6. At last, if no feasible solution is found within $\max_pumps$ iterations, the algorithm terminates with a null solution (line 12, Figure 6).

The original FP framework follows the same algorithm described in Figure 6 with the exception that the decrease rate $\lambda$ given as input is necessarily zero. Then, in the loop of lines 5-11, the $D(x, y, \tilde{x}, \gamma)$ problems are solved under $\gamma = 0$, i.e., without taking into account the original objective function.

7. A Large Neighborhood Search (LNS) heuristic with Path Relinking (PR)

In this section, we describe an LNS heuristic for STOP, which we apply to improve the quality of the initial solutions obtained from the OFP algorithm described in the previous section. The
original LNS metaheuristic framework (Shaw, 1998) works by gradually improving an initial solution through a sequence of destroying and repairing procedures. In our heuristic, the LNS framework is coupled with classical local search procedures widely used to improve solutions of routing problems in general. In particular, these procedures, which are described in Section 7.4, are also present in most of the successful heuristics proposed to solve TOP (e.g., Vansteenwegen et al. (2009); Ke et al. (2008); Souffriau et al. (2010); Kim et al. (2013); Dang et al. (2013b); Ke et al. (2016)).

The heuristic we propose also uses a memory component known as Path Relinking (PR). PR was devised by Glover (1997) and its original version explores a neighborhood defined by the set of intermediate solutions — namely, the “path” — between two given solutions. The PR framework has also been successfully applied to solve TOP (Souffriau et al., 2010).

7.1. Main algorithm

We describe, in Figure 7, the general algorithm of the LNS heuristic we propose. The heuristic receives four inputs: an initial feasible solution — built through OFP of FP —, the number of iterations to be performed (max_iter), the capacity of the pool of solutions (max_pool_size) and a parameter called stalling_limit, which manages how frequently the PR procedure is called. Precisely, it limits the number of iterations in stalling (i.e., with no improvement in the current best solution) before calling the PR procedure. Initially, the variables that keep the current number of iterations (iter_counter) and the number of iterations since the last solution improvement (stalling_counter) are set to zero (line 1, Figure 7). Then, the initial solution Y is improved through local search procedures (detailed in Section 7.4) and added to the initially empty pool \( \Lambda \) (lines 2 and 3, Figure 7).

At this point, an iterative procedure is performed max_iter times (lines 4-27, Figure 7). First, the iteration counter is incremented, and a solution \( Y' \) (randomly selected from the pool) is partially destroyed by the removal of some vertices, as later described in Section 7.2. Then, the algorithm successively tries to improve \( Y' \) (lines 8-15, Figure 7). To this end, the local searches of Section 7.4 are performed on \( Y' \). If the improved \( Y' \) is better than the best solution currently in the pool (i.e., its total profit sum is strictly greater), then the stalling counter is set to -1, and a copy of \( Y' \) is added to \( \Lambda \). The addition of solutions to the pool always considers its capacity. Accordingly, if the pool is not full, the new solution is simply added. Otherwise, it takes the place of the current worst solution in the pool (see lines 10-14, Figure 7).

At this point, the algorithm attempts to do vertex shifting perturbations (line 15, Figure 7), which are detailed in Section 7.5. If it succeeds, the algorithm resumes to another round of local searches. Otherwise, the main loop proceeds by updating the stalling counter. Precisely, if the possibly improved \( Y' \) obtained after the successive local searches and shifting perturbations has greater profit sum than the best solution in \( \Lambda \), then stalling_counter is reset to zero. Otherwise, it is incremented by one (lines 16-18, Figure 7). After that, the algorithm checks if the limit number of iterations in stalling was reached. If that is the case, the PR procedure, whose description is given in Section 7.6, is applied in the current iteration, and the stalling counter is reset once again (lines 19-22, Figure 7).

By the end of the main loop, it is checked if \( Y' \) should be added to \( \Lambda \), which only occurs if the solution does not already belong to the pool and its profit sum is greater than the current worst solution available (lines 23-26, Figure 7). At last, the algorithm returns the best solution in the pool (line 28, Figure 7). In the next sections, we detail all the aforementioned procedures called within the heuristic.
Input: An initial feasible solution \( Y \), \( \text{max}_\text{iter} \in \mathbb{Z}^+ \), \( \text{max}_\text{iter} \geq 1 \), \( \text{max}_\text{pool}_\text{size} \in \mathbb{Z}^+ \), \( \text{max}_\text{pool}_\text{size} \geq 1 \) and \( \text{stalling}_\text{limit} \in \mathbb{Z}^+ \), \( \text{stalling}_\text{limit} \geq 1 \).

Output: An ideally improved feasible solution.

1. Initialize \( \text{iter}_\text{counter} \leftarrow 0 \) and \( \text{stalling}_\text{counter} \leftarrow 0 \);
2. Improve \( Y \) through local searches (see Section 7.4);
3. Initialize the pool of solutions \( \Lambda \leftarrow \{ Y \} \);
4. while (\( \text{iter}_\text{counter} \leq \text{max}_\text{iter} \));
   5. Update \( \text{iter}_\text{counter} \leftarrow \text{iter}_\text{counter} + 1 \);
   6. Randomly select a solution \( Y' \) from \( \Lambda \);
   7. Partially destroy \( Y' \) by removing vertices (see Section 7.2);
   8. do
      9. Improve \( Y' \) through local searches (Section 7.4);
      10. if (\( Y' \) is better than the best solution in \( \Lambda \)) then
          11. \( \text{stalling}_\text{counter} \leftarrow -1 \);
          12. if (|\( \Lambda \)| \( < \text{max}_\text{pool}_\text{size} \)) then \( \Lambda \leftarrow \Lambda \cup \{ Y' \} \);
          13. else Replace the worst solution in \( \Lambda \) with \( Y' \);
      14. end-if;
   15. while (perform inter-route shifting perturbations on \( Y' \)) (Section 7.5);
   16. if (\( Y' \) is better than the best solution in \( \Lambda \)) then
      17. \( \text{stalling}_\text{counter} \leftarrow 0 \);
      18. else \( \text{stalling}_\text{counter} \leftarrow \text{stalling}_\text{counter} + 1 \);
      19. if (\( \text{stalling}_\text{counter} \geq \text{stalling}_\text{limit} \)) then
          20. Perform the PR procedure considering \( \Lambda \) and \( Y' \) (see Section 7.6);
          21. \( \text{stalling}_\text{counter} \leftarrow 0 \);
      22. end-if;
   23. if (\( Y' \notin \Lambda \) and \( Y' \) is better than the worst solution in \( \Lambda \)) then
      24. if (|\( \Lambda \)| \( < \text{max}_\text{pool}_\text{size} \)) then \( \Lambda \leftarrow \Lambda \cup \{ Y' \} \);
      25. else Replace the worst solution in \( \Lambda \) with \( Y' \);
   26. end-if;
   27. end-while;
28. return best solution in \( \Lambda \);

Figure 7: Description of the general LNS algorithm.

7.2. Destroying procedure

The destroying procedure consists of removing some of the profitable vertices belonging to a given feasible solution. Consider a fixed parameter \( \text{removal}_\text{percentage} \in [0, 1] \). First, we determine an upper bound on the number of vertices to be removed, namely \( \text{max}_\text{number}_\text{of}_\text{removals} \). This value is randomly selected in the open interval defined by zero and the product of \( \text{removal}_\text{percentage} \) and the quantity of profitable vertices in the solution. Then, the procedure sequentially performs \( \text{max}_\text{number}_\text{of}_\text{removals} \) attempts of vertex removal, such that, at each time, a visited vertex is randomly selected. Nevertheless, a vertex is only actually removed if it is profitable and the resulting route remains feasible.

7.3. Insertion procedure

Every time the insertion procedure is called, one of two possible priority orders on the unvisited vertices to the inserted is randomly chosen: non-increasing or non-decreasing orders of profits. Then, according to the selected order, the unvisited vertices are individually tested for insertion in the current solution. If a given vertex can be added to the solution (i.e., its addition does not make the routes infeasible), it is inserted in the route and position that increase the least the sum of the
routes’ time durations. Otherwise, the vertex remains unvisited and the next one in the sequence is tested for insertion.

7.4. Local searches

Given a feasible solution, the local searches here adopted attempt to improve the solution quality, which, in this case, means either increasing the profit sum or decreasing the sum of the routes’ times (while maintaining the same profit sum). The general algorithm sequentially performs inter and intra-route improvements, vertex replacements and attempts of vertex insertions, as summarized in Figure 8. The inter and intra-route improvements are detailed in Section 7.4.1 while the vertex replacements are described in Section 7.4.2. The attempts of vertex insertions (lines 3 and 5, Figure 8) are done as described in Section 7.3. The algorithm stops when it reaches a locally optimal solution with respect to the neighborhoods defined by the aforementioned improvement procedures, i.e., no more improvements are achieved.

```
Input: An initial feasible solution Y.
Output: A possibly improved version of Y.
1. do
2. Do inter and intra-route improvements (Section 7.4.1);
3. Try to insert unvisited vertices in Y (Section 7.3);
4. Do vertex replacements (Section 7.4.2);
5. Try to insert unvisited vertices in Y (Section 7.3);
6. while (did any improvement on Y);
7. return Y;
```

Figure 8: Description of the sequence of local searches.

7.4.1. Inter and intra-route improvements

```
Input: An initial feasible solution Y.
Output: A possibly improved version of Y.
1. do
2. // inter-route improvements
3. do
4. for (all combinations of two routes in Y) do
5. Do 1-1 vertex exchange;
6. Do 1-0 vertex exchange;
7. Do 2-1 vertex exchange;
8. end-for;
9. while (did any inter-route improvement);
10. // intra-route improvements
11. for (each route in Y) do
12. Do 3-opt improvement;
13. end-for;
14. while (did any intra-route improvement);
15. return Y;
```

Figure 9: Description of the sequence of the inter and intra-route local searches.
At each iteration of the algorithm of Figure 9 the feasible solution available is first subject to inter-route improvements, i.e., procedures that exchange vertices between different routes (lines 2-9, Figure 9). Precisely, for all combinations of two routes, three kinds of vertex exchanges are performed: (i) 1-1, where a vertex from a route is exchanged with a vertex from another route, (ii) 1-0, where a vertex from a route is moved to another one and (iii) 2-1, where two adjacent vertices from a route are exchanged with a vertex from another route. In the three cases, given a pair of routes, an exchange is only allowed if it preserves the solution’s feasibility and decreases the total sum of the routes’ times. At a call of any of the vertex exchange procedures, the algorithm only performs a single exchange: the first possible by analyzing the routes from beginning to end.

The sequence of inter-route improvements described above is performed until no more exchanges are possible. Then, the algorithm performs the intra-route improvements (lines 10-13, Figure 9). Precisely, for each route of the solution, the classical 3-opt operator (Lin, 1965) is applied. If any improvement is achieved through the 3-opt operator, the algorithm resumes the main loop by performing inter-route improvements once again. Otherwise, it returns the current solution and terminates (line 15, Figure 9).

Without loss of generality, a $k$-opt operator works by repeatedly disconnecting a given route in up to $k$ places and, then, testing for improvement all the possible routes obtained from reconnecting the initially damaged route in all the feasible manners. The operator terminates when no more improvements are possible from removing (and repairing) any combination of $k$ or less arcs. At this point, the route is called $k$-optimal. In Figure 10 we give an example of a round of the 3-opt operator on an arbitrary route of a directed graph. Accordingly, the original route of Figure 10a is
disconnected in three places — identified by dashed arcs —, and reconnected within seven possible manners (Figures 10b and 10c). The three first ones (Figure 10b) are also the moves of the 2-opt operator, as one of the initially removed arcs is always restored (or reversed). In the latter cases (Figure 10c), three of the original arcs are actually discarded. Notice that, depending on the type of reconnection, some arcs of the original route have to be reversed. Thus, in the case of not necessarily complete graphs (such as STOP), some of the rearrangements might not be always feasible.

7.4.2. Vertex replacements

Given a feasible solution, the vertex replacement procedure (Figure 11) works by replacing visited vertices with currently unvisited ones. The selection of unvisited vertices to be inserted in the solution is done exactly as in the insertion procedure described in Section 7.3, i.e., it follows an either non-decreasing or non-increasing order (chosen at random) of vertex profits. The procedure considers two types of replacements, namely 1-1 and 2-1 unvisited vertex exchanges. The former replaces a visited vertex with an unvisited one, while, in the latter, two visited vertices are replaced by an unvisited one. A replacement is only allowed if it preserves the solution’s feasibility and either (i) increases the profit sum or (ii) decreases the sum of the routes’ times while maintaining the same profit sum. At a call of any of the two types of replacements, the algorithm performs a single replacement: the first feasible one found by analyzing the routes from beginning to end.

| Input: An initial feasible solution $Y$. |
| Output: A possibly improved version of $Y$. |
| 1. do |
| 2. Do 1-1 unvisited vertex exchange; |
| 3. Do 2-1 unvisited vertex exchange; |
| 4. while (did any improvement); |
| 5. return $Y$; |

Figure 11: Description of the sequence of vertex replacements.

7.5. Inter-route shifting perturbation

The inter-route shifting perturbation algorithm works by individually testing if the visited vertices can be moved to any other route, as described in Figure 12. Initially, a copy $Y'$ of the initial solution $Y$ is done (line 1, Figure 12). Then, $Y$ is only used as a reference, while all the moves are performed in $Y'$, as to avoid multiple moves of a same vertex. For each route, the algorithm attempts to move each of its vertices (either mandatory or profitable) to a different route in that solution, such that the destination route has the least possible increase in its time duration (lines 3-5, Figure 12). A vertex can be moved to any position of the destination route, as far as both the origin and the destination routes remain feasible. In this case, moves that increase the total sum of the routes’ durations are also allowed.

If at least one vertex is relocated during the attempt to move vertices from a given route, the algorithm proceeds by trying to insert in that route currently unvisited vertices (lines 6-8, Figure 12). In this case, vertices are inserted according to the procedure described in Section 7.3 but only in the route under consideration. After all the originally visited vertices are tested for relocation, $Y$ is replaced by the possibly modified $Y'$ (line 10, Figure 12).
7.6. The PR procedure

The PR procedure works by exploring neighborhoods connecting an initial solution to the ones of a given set of feasible solutions. Here, this set corresponds to the pool of solutions $\Lambda$, which plays the role of a long-term memory. Precisely, at each iteration of the PR procedure, we compute the intermediate solutions — namely, the “path” — between the initial solution and a solution selected from $\Lambda$, as detailed in Figure 13. The algorithm receives as input an initial solution $Y$, the current pool of solutions $\Lambda$, the capacity of $\Lambda$ ($\text{max\_pool\_size}$) and a similarity limit $\epsilon_2 \in [0, 1]$ used to determine which pairs of solutions are eligible to be analyzed.

Initially, the variable that keeps the best solution found so far in the PR procedure ($\text{best\_solution}$) is set to the initial solution $Y$ (line 1, Figure 13). Then, for each solution $X \in \Lambda$, we compute its
similarity with $Y$, which is given by $(2 \times n_{X \cap Y})/(n_X + n_Y)$, where $n_X$ and $n_Y$ stand for the number of vertices visited in the solutions $X$ and $Y$, respectively, and $n_{X \cap Y}$ is the number of vertices common to both solutions. If the similarity metric is inferior to the input limit $\epsilon_2$, the procedure computes the “path” from $Y$ to $X$ and the one from $X$ to $Y$. In both cases, the best solution found in the corresponding “path” (kept in the variable current_solution) is compared with the best solution found so far in the whole PR procedure (best_solution). Then, if applicable (i.e., if the profit sum of current_solution is greater than that of best_solution), best_solution is updated to current_solution. The whole loop described above is summarized at lines 2-14, Figure 13. After that, the algorithm attempts to add best_solution to $\Lambda$ (lines 15-18, Figure 13). Then, the possibly updated pool is returned and the procedure terminates (line 19, Figure 13).

**Input:** A starting solution $Y_s$ and a guiding one $Y_g$.

**Output:** The possibly improved solution.

1. best_solution ← current_solution ← $Y_s$;
2. vertices_to_add ← vertices that are visited in $Y_g$ and not in $Y_s$;
3. Sort vertices_to_add in terms of vertex profits (non-increasing or non-decreasing order);
4. while (vertices_to_add ≠ ∅) do
   5. while (vertices_to_add ≠ ∅) and (there is a feasible route in current_solution) do
      6. Remove a vertex $i$ from vertices_to_add, according to the sorting;
      7. Try to insert $i$ in a still feasible route of current_solution, allowing infeasibility;
   8. end-while;
9. for (each infeasible route $j$ in current_solution) do
   10. Sequentially remove profitable vertices from $j$ to restore its feasibility;
11. end-for;
12. Improve current_solution through local searches (Section 7.4);
13. if (current_solution is better than best_solution) then
14. best_solution ← current_solution;
15. end-if;
16. end-while;
17. return best_solution;

Figure 14: Algorithm for computing the “path” between two solutions.

Given two input feasible solutions — a starting one $Y_s$ and a guiding one $Y_g$ —, the procedure of computing the “path” between them is outlined in Figure 14. Initially, the variables that keep the current and the best solutions found so far in the “path” from $Y_s$ and $Y_g$ are both set to $Y_s$ (line 1, Figure 14). Then, the set of vertices eligible for insertion (namely vertices_to_add) is defined as the vertices that are visited in $Y_g$ and do not belong to $Y_s$ (line 2, Figure 14). Notice that all the vertices in vertices_to_add are profitable, since $Y_s$ and $Y_g$ are both feasible and, thus, visit all the mandatory vertices. After that, vertices_to_add is sorted in non-increasing or non-decreasing order of vertex profits (line 3, Figure 14). This order is chosen at random at each call of the procedure.

Then, while there are eligible vertices in vertices_to_add, the procedure alternates between adding vertices to the current solution — even when it leads to infeasible routes — and removing vertices to restore feasibility. Precisely, at each iteration of the main loop of Figure 14 (lines 4-16), the algorithm attempts to add vertices to current_solution as follows. First, the next vertex in the order established for vertices_to_add is removed from the set. Then, if possible, such vertex is inserted in the route and position that increase the least the sum of the routes’ time durations. In this case, an insertion that makes a route infeasible in terms of time limit is also allowed.
Nevertheless, once a route becomes infeasible, it is no longer considered for further insertions. The rounds of insertion continue until either `vertices_to_add` gets empty or all the routes of `current_solution` become infeasible (see lines 5-8, Figure 14).

At this point, the algorithm removes profitable vertices from `current_solution` to restore its feasibility. In particular, for each infeasible route, profitable vertices are sequentially removed in non-decreasing order of its profits until all the routes become feasible again (lines 9-11, Figure 14). If the removal of a vertex would disconnect the route it belongs, then this vertex is preserved. In these cases, the next candidate vertex is considered for removal, and so on.

After the removals, the algorithm attempts to improve the now feasible `current_solution` through the local searches described in Section 7.4 (line 12, Figure 14), and, if applicable, `best_solution` is updated to `current_solution` (lines 13-15, Figure 14). After the main loop terminates, `best_solution` is returned (line 17, Figure 14).

8. Implementation details

All the codes were developed in C++, and the LP problems that arise in the cutting-plane used to reinforce formulation $\mathcal{F}$ were solved by means of the optimization solver ILOG CPLEX 12.4.1. We kept the default configurations of CPLEX in our implementation.

In the sequel, we describe some specific implementation choices for the separation of cuts, along with the parameter configurations adopted in each algorithm. These configurations were established according to previous studies in the literature, as well as to pilot tests on a control set of 10 STOP instances, composed of both challenging instances and some of the smallest ones. This control set is detailed in Table 1, where we report, for each instance, the number of vertices ($|N|$), the number of vehicles ($|M|$) and the route duration limit ($T$). The reduced number of instances was chosen as a way to prevent the parameter configuration of the heuristic from being biased by the control set (overfitting). The whole benchmark of instances adopted in this study is later detailed in Section 9.1.

| Instance     | $|N|$ | $|M|$ | $T$  |
|--------------|------|------|------|
| p3.3.r_5%    | 33   | 3    | 33.3 |
| p4.3.j_5%    | 100  | 3    | 46.7 |
| p4.3.n_5%    | 100  | 3    | 60.0 |
| p5.3.m_5%    | 66   | 3    | 21.7 |
| p5.3.r_5%    | 66   | 3    | 30.0 |
| p6.2.k_5%    | 64   | 2    | 32.5 |
| p6.3.m_5%    | 64   | 3    | 25.0 |
| p6.3.n_5%    | 64   | 3    | 26.7 |
| p7.3.o_5%    | 102  | 3    | 100.0|
| p7.3.p_5%    | 102  | 3    | 106.7|

8.1. Cut separation

Regarding the separation of CCCs, we solved maximum flow sub-problems with the implementation of the preflow push-relabel algorithm of Goldberg and Tarjan (1988) provided by the

---

1[^1]: [http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/](http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/)
open-source Library for Efficient Modeling and Optimization in Networks — LEMON (Dezső et al., 2011). The knapsack sub-problems that arise during the separation of LCIs were solved through classical dynamic programming based on Bellman recursion (Bellman, 1957).

While selecting cuts, we adopted the absolute violation criterion to determine which inequalities are violated by a given solution and the so-called distance criterion to properly compare two cuts, i.e., to determine which one is most violated by a solution. Given an n-dimensional column vector \( w \) of binary variables, a point \( \bar{w} \in \mathbb{R}^n \) and an inequality of the general form \( a^T w \leq b \), with \( a \in \mathbb{R}^n \), \( b \in \mathbb{R} \), the absolute violation of this inequality with respect to \( \bar{w} \) is simply given by \( a^T \bar{w} - b \).

Moreover, the distance from \( a^T w \leq b \) to \( \bar{w} \) corresponds to the Euclidean distance between the hyperplane \( a^T w = b \) and \( \bar{w} \), which is equal to \( \frac{(a^T \bar{w} - b)}{\|a\|} \), where \( \|a\| \) is the Euclidean norm of \( a \).

We set two parameters for each type of inequality separated in the cutting-plane algorithm: a precision one used to classify the inequalities into violated or not (namely absolute violation precision), and another one to discard cuts that are not sufficiently orthogonal to the most violated ones. The latter parameter determines the minimum angle that an arbitrary cut must form with the most violated cut, as not to be discarded. In practice, this parameter establishes the maximum acceptable inner product between the arbitrary cut and the most violated one. Accordingly, we call it the maximum inner product. In the case of two inequalities \( a_1^T w \leq b_1 \) and \( a_2^T w \leq b_2 \), with \( a_1, a_2 \in \mathbb{R}^n \) and \( b_1, b_2 \in \mathbb{R} \), the inner product between them is given by \( \frac{(a_1^T a_2)}{\|a_1\| \|a_2\|} \) and corresponds to the cosine of the angle defined by them. Since AVICs are separated by complete enumeration, the aforementioned parameters do not apply to them. The values adopted for these parameters are shown in Table 2. The tolerance input value \( \epsilon_1 \) of the cutting-plane algorithm was set to \( 10^{-3} \).

Table 2: Parameter configuration adopted in the separation and selection of valid inequalities in the cutting-plane used to reinforce \( \mathcal{F} \).

| Inequalities | Absolute violation precision | Maximum inner product |
|--------------|-----------------------------|-----------------------|
| CCCs         | 0.01                        | 0.03                  |
| AVICs        | –                           | –                     |
| LCIs         | \( 10^{-5} \)               | –                     |

8.2. Parameter configuration adopted for the FP and the LNS heuristics

In Tables 3 and 4, we summarize the values adopted for the input parameters of the FP framework described in Figure 6 and the LNS heuristic, respectively. In the latter table, we include the parameters related to the procedures called within the main algorithm described in Figure 7.

Table 3: Parameter configuration adopted for the FP heuristic.

| Parameter | \( \text{max\_pumps} \) | \( \lambda \) | \( K \) |
|-----------|--------------------------|---------------|--------|
| Value     | 2000                     | 0.9           | 10     |

9. Computational experiments

The computational experiments were performed on a 64 bits Intel Core i7-4790K machine with 4.0 GHz and 15.0 GB of RAM, under Linux operating system. The machine has four physical
Table 4: Parameter configuration adopted for the LNS heuristic.

| Parameter                  | max_iter | max_pool_size | stalling_limit | removal_percentage | $\epsilon$ |
|----------------------------|----------|---------------|----------------|--------------------|-----------|
| Value                      | \{1000, 2000, 5000\} | 20            | 100            | 0.75               | 0.9       |

cores, each one running at most two threads in hyper-threading mode. In our experiments, we tested several variations of the LNS heuristic obtained by considering FP and OFP, the original formulation $\mathcal{F}$ and its reinforced version, as well as different numbers of iterations for the main LNS algorithm. These variations are later detailed in Sections 9.4 and 9.5.

9.1. Benchmark instances

In our experiments, we used the benchmark of instances adopted by Assunção and Mateus (2019), which consists of complete graphs with up to 102 vertices. These instances were generated based on the TOP ones of Chao et al. (1996) by randomly selecting vertices to be mandatory. In each instance, only a small percentage of the vertices (5%) is set as mandatory, as to avoid infeasibility. The 387 instances in the benchmark are divided into seven data sets, according to the number of vertices of their graphs. Within a given data set, the instances only differ by the time limit imposed on the route duration and the number of vehicles. The characteristics of the seven data sets are detailed in table 5. For each set, it is reported the number of instances (#), the number of vertices in the graphs ($|N|$), the possible numbers of vehicles available ($|M|$) and the range of values that the route duration limit $T$ assumes.

Table 5: Description of the benchmark of STOP instances adopted in the experiments.

| Set     | 1_5% | 2_5% | 3_5% | 4_5% | 5_5% | 6_5% | 7_5% |
|---------|------|------|------|------|------|------|------|
| #       | 54   | 33   | 60   | 60   | 78   | 42   | 60   |
| $|N|$    | 32   | 21   | 33   | 100  | 66   | 64   | 102  |
| $|M|$    | 2–4  | 2–4  | 2–4  | 2–4  | 2–4  | 2–4  | 2–4  |
| $T$     | 3.8–22.5 | 1.2–42.5 | 3.8–55 | 3.8–40 | 1.2–65 | 5–200 | 12.5–120 |

According to previous experiments (Assunção and Mateus, 2019), instances whose graphs have greater dimensions are the hardest (sets 4_5%, 5_5% and 7_5%) to be solved to optimality. Moreover, within a same set, instances with greater route duration limits (given by $T$) also tend to be more difficult. This is possibly due to the fact that greater limits imply more feasible routes, thus increasing the search space.

In this work, we pre-processed all the instances by removing vertices and arcs that are inaccessible with respect to the limit $T$ imposed on the total traverse times of the routes. To this end, we considered the $R$ matrix defined in the beginning of Section 4, which keeps, for each pair of vertices, the time duration of a minimum time path between them. Moreover, we deleted all the arcs that either enter the origin $s$ or leave the destination $t$, as to implicitly satisfy constraints (5). Naturally, the time spent in these pre-processings are included in the execution times of the algorithms tested.

9.2. Statistical analysis adopted

Since all the heuristic algorithms proposed have randomized choices within their execution, we ran each algorithm 10 times for each instance to properly assess their performance. In this sense,
we considered a unique set of 10 seeds common to all algorithms and instances tested. To evaluate the quality of the solutions obtained by the heuristics proposed, we compared them with the best known primal solutions/bounds provided by the cutting-plane algorithm of Assunção and Mateus (2019).

To assess the statistical significance of the results, we follow the tests suggested by Demšar (2006) for the simultaneous comparison of multiple algorithms on different data (instance) sets. Precisely, we first apply the Iman-Davenport test (Iman and Davenport, 1980) to check the so-called null hypothesis, i.e., the occurrence of equal performances with respect to a given indicator (e.g., quality of the solution’s bounds). If the null hypothesis is rejected, and, thus, the algorithms’ performances differ in a statistically significant way, a post-hoc test is performed to analyze these differences more closely. In our study, we adopt the post-hoc test proposed by Nemenyi (1963).

The Iman-Davenport test is a more accurate version of the non-parametric Friedman test (Friedman, 1937). In both tests, the algorithms considered are ranked according to an indicator of performance. Let $I$ be the set of instances and $J$ be the set of algorithms considered. In our case, the algorithms are ranked for each instance separately, such that $r_{ji}$ stands for the rank of an algorithm $j \in J$ while solving an instance $i \in I$. Accordingly, the value of $r_{ji}$ lies within the interval $[1, |J|]$, such that better performances are linked to smaller ranks (in case of ties, average ranks are assigned). The average ranks (over all instances) of the algorithms — given by $R_j = \frac{1}{|I|} \sum_{i \in I} r_{ji}$ for all $j \in J$ — are used to compute a statistic $F_F$, which follows the F distribution (Sheskin, 2007).

Critical values are determined considering a significance level $\alpha$, which, in this case, indicates the probability of the null hypothesis being erroneously rejected. In practical terms, the smaller $\alpha$, the greater the statistical confidence of the test. Accordingly, in the case of the Iman-Davenport test, the null hypothesis is rejected if the statistic $F_F$ is greater than the critical value. In our experiments, we alternatively test the null hypothesis by determining (through a statistical computing software) the so-called $p$-value, which provides the smallest level of significance at which the null hypothesis would be rejected. In other words, given an appropriate significance level $\alpha$ (usually, at most 5%), we can safely discard the null hypothesis if $p$-value $\leq \alpha$.

Once the null hypothesis is rejected, we can apply the post-hoc test of Nemenyi (1963), which compares the algorithms in a pairwise manner. The performances of two algorithms $j, k \in J$ are significantly different if the corresponding average ranks $R_j$ and $R_k$ differ by at least a Critical Difference (CD). In our experiments, we used the R open software for statistical computing to compute all of the statistics needed, including the average ranks and the CDs.

9.3. Bound improvement provided by the new valid inequalities

We first analyzed the impact of the new inequalities proposed in Section 4.1 on the strength of the formulation $\mathcal{F}$. To this end, we computed the dual (upper) bounds obtained from adding these inequalities to $\mathcal{L}$ (the linear relaxation of $\mathcal{F}$) according to 10 different configurations, as described in Table 6. To ease comparisons, we also display the results for previous inequalities proposed in the literature, namely the General Connectivity Constraints (GCCs) of Bianchessi et al. (2018) and the Conflict Cuts (CCs) of Assunção and Mateus (2019). For each instance and configuration, we solved the cutting-plane algorithm described in Section 5 while considering only the types of inequalities of the corresponding configuration. For the configurations including GCCs and CCs, we used, within the cutting-plane algorithm, the separation procedures adopted by Assunção and Mateus (2019). In all these experiments, the CPLEX built-in cuts are disabled.

\[ \text{https://www.r-project.org/} \]
Table 6: Configurations of valid inequalities. GCCs and CCs stand for the General Connectivity Constraints of Bianchessi et al. (2018) and the Conflict Cuts (CCs) of Assunção and Mateus (2019), respectively.

| Configuration | GCCs | CCs | CCCs | LCIs | AVICs |
|---------------|------|-----|------|------|-------|
| 1             | ×    |     |      |      |       |
| 2             | ×    |     |      |      |       |
| 3             |     |     |      |      |       |
| 4             |     |     |      |      |       |
| 5             |     |     |      |      |       |
| 6             | ×    | ×   |      |      |       |
| 7             | ×    | ×   |      |      |       |
| 8             | ×    | ×   |      |      |       |
| 9             |     | ×   |      |      |       |
| 10            |     |     |      |      |       |

The results obtained are detailed in Table 7. The first column displays the name of each instance set. Then, for each configuration of inequalities, we give the average and the standard deviation (over all the instances in each set) of the percentage bound improvements obtained from the addition of the corresponding inequalities. Without loss of generality, given an instance, its percentage improvement in a configuration $i \in \{1, \ldots, 10\}$ is given by $100 \cdot \frac{UB_{LP} - UB_i}{UB_{LP}}$, where $UB_{LP}$ denotes the bound provided by $\mathcal{L}$, and $UB_i$ stands for the bound obtained from solving the cutting-plane algorithm of Section 5 in the configuration $i$. The last row displays the numerical results while considering the whole benchmark of instances.
Table 7: Percentage dual (upper) bound improvements obtained from adding to $\mathcal{L}$ the inequalities of Section 4.1 according to the 10 configurations in Table 6.

| Configuration of inequalities | \(1\) — GCCs | \(2\) — CCs | \(3\) — CCCs | \(4\) — LCIs | \(5\) — AVICs |
|-------------------------------|-------------|-------------|-------------|-------------|-------------|
| Set                           | Avg (%)     | StDev (%)   | Avg (%)     | StDev (%)   | Avg (%)     | StDev (%)   | Avg (%)     | StDev (%)   | Avg (%)     | StDev (%)   |
| \(1\_5\%\)                   | 7.32        | 4.67        | 8.83        | 5.98        | 8.88        | 4.09        | 1.00        | 3.93        | 5.05        | 3.41        |
| \(2\_5\%\)                   | 0.32        | 0.90        | 0.86        | 2.44        | 0.64        | 1.81        | 0.51        | 1.43        | 0.22        | 0.61        |
| \(3\_5\%\)                   | 1.48        | 1.35        | 2.71        | 1.90        | 3.79        | 2.60        | 0.59        | 0.99        | 1.39        | 1.14        |
| \(4\_5\%\)                   | 5.94        | 5.62        | 5.15        | 6.46        | 8.80        | 7.33        | 0.00        | 0.01        | 5.52        | 5.01        |
| \(5\_5\%\)                   | 0.18        | 0.65        | 0.86        | 1.46        | 1.10        | 2.01        | 0.02        | 0.10        | 0.22        | 0.43        |
| \(6\_5\%\)                   | 0.06        | 0.17        | 0.69        | 2.24        | 0.18        | 0.45        | 0.03        | 0.07        | 0.18        | 0.53        |
| \(7\_5\%\)                   | 5.96        | 4.24        | 8.95        | 9.58        | 9.69        | 4.77        | 0.00        | 0.00        | 7.30        | 5.97        |
| Total                         | 3.18        | 4.46        | 4.03        | 5.95        | 5.01        | 5.59        | 0.26        | 1.52        | 3.00        | 4.31        |

| Configuration of inequalities | \(6\) — GCCs & CCs | \(7\) — 6 & LCIs | \(8\) — 7 & AVICs | \(9\) — CCCs & LCIs | \(10\) — 9 & AVICs |
|-------------------------------|-------------------|-----------------|------------------|-------------------|------------------|
| Set                           | Avg (%)           | StDev (%)       | Avg (%)          | StDev (%)         | Avg (%)          | StDev (%)       | Avg (%)          | StDev (%)         | Avg (%)          | StDev (%)         |
| \(1\_5\%\)                   | 9.26              | 6.01            | 9.46             | 6.00              | 9.49             | 6.05              | 9.77             | 6.65              | 9.13             | 4.01              |
| \(2\_5\%\)                   | 0.85              | 2.41            | 0.97             | 2.75              | 1.00             | 2.82              | 1.12             | 3.18              | 1.21             | 3.41              |
| \(3\_5\%\)                   | 2.71              | 1.87            | 3.22             | 2.05              | 3.28             | 2.04              | 4.00             | 2.73              | 4.07             | 2.80              |
| \(4\_5\%\)                   | 7.27              | 6.62            | 7.27             | 6.63              | 7.52             | 6.58              | 8.83             | 7.31              | 10.00            | 9.77              |
| \(5\_5\%\)                   | 0.84              | 1.32            | 0.87             | 1.40              | 0.88             | 1.39              | 1.08             | 1.97              | 1.25             | 2.08              |
| \(6\_5\%\)                   | 0.70              | 2.24            | 0.73             | 2.23              | 0.74             | 2.24              | 0.21             | 0.45              | 0.28             | 0.55              |
| \(7\_5\%\)                   | 10.16             | 8.85            | 10.08            | 8.59              | 10.43            | 8.60              | 9.55             | 4.70              | 10.63            | 5.41              |
| Total                         | 4.63              | 6.14            | 4.75             | 6.09              | 4.86             | 6.14              | 5.19             | 5.97              | 5.52             | 6.56              |
Table 8: Average number of cuts separated by the cutting-plane algorithm running under configuration 10. AVICs were omitted, as they are separated by complete enumeration.

| Set   | LCIs | CCCs |
|-------|------|------|
| 1._5% | 1.78 | 13.26|
| 2._5% | 0.33 | 1.00 |
| 3._5% | 3.12 | 10.80|
| 4._5% | 1.02 | 88.57|
| 5._5% | 0.67 | 17.41|
| 6._5% | 0.29 | 0.71 |
| 7._5% | 0.38 | 68.28|
| **Total** | **1.14** | **31.51** |

The results exposed in Table 7 indicate that, on average, CCCs are the inequalities that strengthen formulation $F$ the most, followed by CCs, GCCs, AVICs and LCIs. The results also show that coupling CCCs with LCIs and AVICs gives the best average bound improvement (5.52%) among all the configurations of inequalities tested.

One may notice that, for some instance sets, coupling different types of inequalities gives worse average bound improvements than considering only a subset of them (see, e.g., set 5._5% under configuration 2 and 6, Table 7). Also notice that, although CCCs dominate GCCs and CCs (Corollary 8), there are cases where CCCs give worse average bound improvements than CCs (see set 2._5%, Table 7). Both behaviours can be explained by the fact that these inequalities are separated heuristically, as the separation procedures only consider the cuts that are violated by at least a constant factor. Moreover, they adopt a stopping condition based on bound improvement of subsequent iterations, which might halt the separation before all the violated cuts are found. Then, in practical terms, the separation algorithms adopted do not necessarily give the actual theoretical bounds obtained from the addition of the inequalities proposed.

For completeness, we display, in Table 8 the average number of cuts separated for each class of inequalities when considering configuration 10 — the one adopted in our FP algorithms. We omitted the average number of AVICs, as they are separated by complete enumeration and the number of these cuts in each instance always corresponds to $2 \times |E|$.

9.4. Results for the FP algorithms

We first compared four variations of the FP framework discussed in Section 6, as summarized in Table 9. Precisely, we considered both FP and OFP while based on the original formulation $F$ and the one reinforced according to the cutting-plane algorithm described in Section 5. The latter formulation is referred to as $F+cuts$.

In Table 10 we report the results obtained by the four FP algorithms described in Table 9. For each algorithm and instance set, we report four values: (i) the average and (ii) the standard deviation of the relative gaps given by $100 \cdot \frac{LB^* - LB}{LB}$, where $LB^*$ is the best known lower (primal) bound for the instance (provided by the exact algorithm of Assunção and Mateus (2019)), and $LB$ is the average bound (over the 10 executions) obtained by the corresponding FP algorithm; (iii) the average number of iterations/pumps (over the 10 executions and all instances of the set), and (iv) the average wall-clock processing time (in seconds). We highlight that, for the cases where a feasible solution is not found in any of the 10 executions, the corresponding relative gaps of (i) are set to 100%. The same is done in case no primal bound is available from the baseline exact
Table 9: Variations of FP analyzed in our study. $\mathcal{F}+$-cuts stands for the reinforced version of the formulation $\mathcal{F}$ discussed in Section 5.

| Algorithm   | Framework | Base formulation |
|-------------|-----------|------------------|
|             | FP        | OFP              | $\mathcal{F}$ | $\mathcal{F}+$-cuts |
| FP_raw      | x         | x                |              |                   |
| FP_cuts     | x         |                   |              |                   |
| OFP_raw     | x         | x                |              |                   |
| OFP_cuts    | x         |                   |              |                   |

Moreover, when the instance is proven to be infeasible, the gap is set to 0%, as well as when $LB^* = LB = 0$. Entries in bold discriminate the cases where the reinforcement of the original formulation led to better performances or not.

Table 10: Summary of the results obtained by the four FP algorithms described in Table 9. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and number of pumps.

| Set | FP_raw | | | | FP_cuts | | | | | | OFP_raw | | | | | OFP_cuts |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|     | Gap (%) | Avg | StDev | Pumps (#) | Time (s) | Gap (%) | Avg | StDev | Pumps (#) | Time (s) | Gap (%) | Avg | StDev | Pumps (#) | Time (s) | Gap (%) | Avg | StDev | Pumps (#) | Time (s) |
| 1_5% | 26.60 | 29.92 | 5.86 | 0.06 | 26.19 | 29.58 | 4.59 | 0.10 | 23.80 | 26.71 | 18.09 | 0.44 | 11.18 | 22.00 | 8.07 | 0.02 | 15.24 | 23.20 | 90.91 | 17.56 |
| 2_5% | 11.74 | 23.61 | 2.19 | 0.00 | 9.18 | 20.82 | 1.75 | 0.00 | 11.18 | 22.00 | 8.07 | 0.02 | 11.18 | 22.00 | 8.07 | 0.02 | 15.24 | 23.20 | 90.91 | 17.56 |
| 3_5% | 29.47 | 26.30 | 7.32 | 0.08 | 28.24 | 24.33 | 6.12 | 0.14 | 18.49 | 19.70 | 53.41 | 56.76 | 16.42 | 24.44 | 53.40 | 83.76 |
| 4_5% | 29.69 | 24.62 | 86.59 | 39.38 | 18.49 | 19.70 | 53.41 | 56.76 | 16.42 | 24.44 | 53.40 | 83.76 | 16.42 | 24.44 | 53.40 | 83.76 | 16.42 | 24.44 | 53.40 | 83.76 |
| 5_5% | 27.76 | 26.33 | 38.06 | 3.33 | 24.25 | 25.67 | 34.91 | 10.96 | 24.25 | 25.67 | 34.91 | 10.96 | 24.25 | 25.67 | 34.91 | 10.96 | 24.25 | 25.67 | 34.91 | 10.96 |
| 6_5% | 12.46 | 16.87 | 5.23 | 0.45 | 13.66 | 18.07 | 5.05 | 0.70 | 13.66 | 18.07 | 5.05 | 0.70 | 13.66 | 18.07 | 5.05 | 0.70 | 13.66 | 18.07 | 5.05 | 0.70 |
| 7_5% | 27.66 | 34.21 | 161.26 | 109.12 | 16.42 | 24.44 | 53.40 | 83.76 | 16.42 | 24.44 | 53.40 | 83.76 | 16.42 | 24.44 | 53.40 | 83.76 | 16.42 | 24.44 | 53.40 | 83.76 |
| Total | 25.12 | 27.51 | 48.80 | 23.77 | 20.60 | 24.48 | 25.88 | 24.11 | 23.80 | 26.71 | 18.09 | 0.44 | 11.18 | 22.00 | 8.07 | 0.02 | 15.24 | 23.20 | 90.91 | 17.56 |

First, notice that, in terms of solution quality, the results of the four algorithms are rather poor, as the average relative gaps and standard deviations are quite high. Nevertheless, these results do not indicate unsuccessful performances, as the general FP framework was devised to find feasible...
solutions, usually at the expense of solution quality. As expected, the quality of the solutions obtained by the algorithms that use the OFP framework is slightly superior (on average) to that of solutions obtained by the algorithms based on pure FP. Nevertheless, as we will discuss later, we cannot assure that the gain of OFP is statistically significant in this case.

More importantly, the results suggest that, on average, the reinforcement of the original formulation $F$ improves both the quality of the solutions obtained and the convergence (signaled by the number of pumps) of the algorithms to feasible solutions. In particular, for all instance sets but 6.5%, the average gaps of the solutions obtained by FP_cuts are strictly smaller (and, thus, better) than those obtained by FP_raw. In the case of the average numbers of pumps, FP_cuts outperforms FP_raw for all instance sets. When comparing OFP_cuts and OFP_raw, the same pattern is verified. Regarding average execution times, the four algorithms are comparable.

We applied the Iman-Davenport and the Nemenyi tests to validate the statistical significance of the results discussed above. While comparing the relative gaps, we ranked the algorithms according to the values $100 \cdot \left(1 - \frac{LB^* - LB}{LB^*}\right)$, as to make smaller gaps imply greater ranks. Similarly, while comparing the number of pumps, the algorithms were ranked based on the maximum number of pumps (parameter max_iter of Figure 6) minus the actual number of pumps performed. With respect to the relative gaps, we obtained the statistic $F_F = 10.76$, with $p$-value $= 5.62 \cdot 10^{-7}$. Regarding the number of pumps, we obtained the statistic $F_F = 94.06$, with $p$-value $= 2.2 \cdot 10^{-16}$. Then, in both cases, we can safely reject the null hypothesis of equal performances.

To compare the performance of the algorithms in a pairwise manner, we proceeded with the Nemenyi test. Figure 15 depicts the average ranks of the four FP algorithms and the Critical Difference (CD) while considering a significance level $\alpha = 5\%$. Connections between algorithms indicate non-significant differences, i.e., the difference between the corresponding pair of average ranks is no greater than the CD. Figures 15a and 15b are based on the relative gaps and the number of pumps, respectively.

With respect to the relative gaps, we can conclude that the performance of OFP_cuts is significantly better than those of OFP_raw and FP_raw. Moreover, FP_cuts outperforms FP_raw. Nevertheless, we cannot conclude if OFP_cuts is significantly better than FP_cuts, neither that OFP_raw is better than FP_raw in this same indicator. Regarding the convergence (number of pumps), FP_cuts outperforms all the other algorithms. In summary, from the results, we can safely conclude that the reinforcement of $F$ here applied yields better performances both in terms of convergence and solution quality. In addition, considering both indicators, FP_cuts stands out as the best option.

9.5. Results for the LNS heuristic

In the experiments concerning the LNS heuristic, we first tested the impact of the number of iterations on the quality of the final solutions obtained. To this end, we tested six variations of the heuristic by considering the two best FP algorithms — FP_cuts and OFP_cuts — coupled with the LNS framework of Section 7 running for max_iter $\in \{1000, 2000, 5000\}$ iterations. Hereafter, the variations of our heuristic are named (O)FP_cuts_LNS_max_iter. Accordingly, without loss of generality, an algorithm that receives an initial solution provided by FP_cuts and runs the LNS heuristic for 1000 iterations is referred to as FP_cuts_LNS_1000.

In Table 11 we summarize the results obtained by the six LNS algorithms tested. For each algorithm and instance set, we report five values: (i) the average and (ii) the standard deviation of the relative gaps given by $100 \cdot \frac{LB^* - LB}{LB^*}$, where LB is the average bound (over the 10 executions) obtained by the corresponding algorithm. Recall that LB* is the best known bound for the instance (provided by the exact algorithm of Assunção and Mateus (2019)); (iii) the average (over the 10
Figure 15: FP algorithms’ average ranks depicted on number lines, along with the Critical Difference (CD), when considering a significance level $\alpha = 5\%$. Connections between algorithms indicate non-significant differences.

The results indicate that, on average, the LNS algorithms that use the initial solutions provided
Table 11: Summary of the results obtained by six variations of the LNS algorithm tested. The execution times do not consider the time spent by the FP algorithms in finding initial solutions. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and profit sums.

| Set  | FP_cuts_LNS_1000 | OFP_cuts_LNS_1000 |
|------|-----------------|-------------------|
|      | Gap (%) Profit sum | Gap (%) Profit sum | Time (s) |
|      | Avg StDev Avg Best | Avg StDev Avg Best | Avg StDev Avg Best |
| 1_5% | 0.37 1.91 79.34 79.63 | 0.00 0.00 79.63 79.63 | 0.10 |
| 2_5% | 0.00 0.00 41.52 41.52 | 0.00 0.00 41.52 41.52 | 0.01 |
| 3_5% | 0.00 0.00 308.50 308.50 | 0.00 0.00 308.50 308.50 | 0.20 |
| 4_5% | -0.07 1.22 609.37 613.75 | -0.19 1.08 612.15 614.23 | 7.10 |
| 5_5% | 1.34 11.32 602.36 603.14 | 1.34 11.32 602.40 603.21 | 1.04 |
| 6_5% | 0.01 0.03 375.20 375.29 | 0.02 0.05 375.11 375.29 | 0.55 |
| 7_5% | 1.79 12.90 306.32 311.22 | 3.48 18.08 292.82 295.20 | 1.36 |
| Total | 0.59 7.23 366.53 368.18 | 0.78 8.79 364.91 365.78 | 1.63 |

| Set  | FP_cuts_LNS_2000 | OFP_cuts_LNS_2000 |
|------|-----------------|-------------------|
|      | Gap (%) Profit sum | Gap (%) Profit sum | Time (s) |
|      | Avg StDev Avg Best | Avg StDev Avg Best | Avg StDev Avg Best |
| 1_5% | 0.37 1.91 79.34 79.63 | 0.00 0.00 79.63 79.63 | 0.20 |
| 2_5% | 0.00 0.00 41.52 41.52 | 0.00 0.00 41.52 41.52 | 0.03 |
| 3_5% | 0.00 0.00 308.50 308.50 | 0.00 0.00 308.50 308.50 | 0.39 |
| 4_5% | -0.15 1.19 610.07 614.02 | -0.25 1.11 612.78 614.30 | 14.75 |
| 5_5% | 1.31 11.32 602.70 603.14 | 1.30 11.32 602.74 603.21 | 2.08 |
| 6_5% | 0.00 0.02 375.24 375.29 | 0.01 0.03 375.21 375.29 | 1.09 |
| 7_5% | 1.74 12.90 306.73 311.28 | 3.44 18.08 292.13 295.25 | 2.85 |
| Total | 0.56 7.23 366.78 368.23 | 0.76 8.80 365.14 365.80 | 3.30 |

| Set  | FP_cuts_LNS_5000 | OFP_cuts_LNS_5000 |
|------|-----------------|-------------------|
|      | Gap (%) Profit sum | Gap (%) Profit sum | Time (s) |
|      | Avg StDev Avg Best | Avg StDev Avg Best | Avg StDev Avg Best |
| 1_5% | 0.37 1.91 79.34 79.63 | 0.00 0.00 79.63 79.63 | 0.50 |
| 2_5% | 0.00 0.00 41.52 41.52 | 0.00 0.00 41.52 41.52 | 0.07 |
| 3_5% | 0.00 0.00 308.50 308.50 | 0.00 0.00 308.50 308.50 | 0.98 |
| 4_5% | -0.24 1.17 611.07 614.47 | -0.30 1.14 613.28 614.43 | 37.46 |
| 5_5% | 1.29 11.32 602.96 603.21 | 1.29 11.32 602.94 603.21 | 5.30 |
| 6_5% | 0.00 0.01 375.26 375.29 | 0.00 0.02 375.26 375.29 | 2.74 |
| 7_5% | 1.72 12.90 306.92 311.30 | 3.40 18.09 293.45 295.27 | 7.50 |
| Total | 0.54 7.23 367.02 368.32 | 0.74 8.80 365.31 365.82 | 8.29 |

by FP_cuts and OFP_cuts reach final solutions with comparable quality. In addition, as expected, increasing the number of iterations does improve (on average) the quality of the solutions obtained. Also notice that the average execution times are almost the same for algorithms with a same number
of iterations, which is in accordance with the stopping criterion adopted in the algorithms. In fact, the LNS heuristics run fairly fast, being the search for initial feasible solutions responsible for the majority of the computational effort (see again Table 10). Such behaviour was expected, since finding an initial solution for STOP is an NP-hard task (Corollary 1).

Notice that, for the instance set with the greatest dimensions (set 7_5%), the standard deviation of the relative gaps obtained by the six algorithms were particularly high. This is partially due to the fact that, for a few instances in this set, the cutting-plane baseline could neither find feasible solutions nor prove their infeasibility. In these cases, the relative gaps were set to 100%.

As for the FP heuristics, we also applied the Iman-Davenport and the Nemenyi tests to validate the statistical significance of the results discussed above. We highlight that, although better (greater) profit sums imply smaller relative gaps on a per-instance basis, the same does not hold when we consider average values, since relative gaps are normalized by definition. Once our statistical tests use the results on a per-instance basis, considering either relative gaps or profit sums leads to a same ranking of the algorithms. In particular, we ranked the algorithms according to the values

$$100 \cdot \left(1 - \frac{LB^* - LB}{LB^*}\right),$$

as to make smaller gaps imply greater ranks.

![Critical Difference (CD)](image)

Figure 16: LNS algorithms’ average ranks depicted on a number line, along with the Critical Difference (CD), when considering a significance level $\alpha = 5\%$. Connections between algorithms indicate non-significant differences.

Regarding the Iman-Davenport test, we obtained the statistic $F_F = 7.26$, with $p$-value $= 9.49 \cdot 10^{-7}$, thus rejecting the null hypothesis of equal performances of all the six algorithms tested. Then, proceeding with the Nemenyi test, Figure 16 depicts the average ranks of the six LNS algorithms and the Critical Difference (CD) while considering a significance level $\alpha = 5\%$. Connections between algorithms indicate non-significant differences with respect to CD. From the results, we cannot conclude if the LNS algorithms based on FP_cuts differ significantly (in terms of the quality of the solutions obtained) from the corresponding ones based on OFP_cuts. On the other hand, the results clearly indicate that increasing the number of iterations from 1000 to 5000 does lead to a statistically significant improvement.

In Table 12, we summarize the number of instances for which the two best variations of the heuristic (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) were able to reach the best previously known bounds. For each algorithm and instance set, we also indicate the number of cases where the LNS algorithm improved over the best known solutions.

At last, we tested if dramatically increasing the number of iterations (to 100000) could close
Table 12: Number of times the two best algorithms (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) reached and/or improved over the best known bounds. Bold entries highlight, for each instance set, the algorithm(s) that performed best.

| Set | FP_cuts_LNS_5000 Reached best (#) | Improved best (#) | OFP_cuts_LNS_5000 Reached best (#) | Improved best (#) |
|-----|----------------------------------|-------------------|----------------------------------|-------------------|
| 1_5%| 54/54                            | 0/54              | 54/54                            | 0/54              |
| 2_5%| 33/33                            | 0/33              | 33/33                            | 0/33              |
| 3_5%| 60/60                            | 0/60              | 60/60                            | 0/60              |
| 4_5%| 59/60                            | 18/60             | 59/60                            | 17/60             |
| 5_5%| 76/78                            | 3/78              | 76/78                            | 3/78              |
| 6_5%| 42/42                            | 0/42              | 42/42                            | 0/42              |
| 7_5%| 58/60                            | 0/60              | 54/60                            | 0/60              |
| **Total** | **382/387** | **21/387** | **378/387** | **20/387** |

Table 13: Summary of the results obtained by FP_cuts_LNS_100000 and OFP_cuts_LNS_100000 while solving two of the more challenging instance sets. The execution times do not consider the time spent by the FP algorithms in finding initial solutions. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and profit sums.

| Set | FP_cuts_LNS_100000 Gap (%) | Profit sum | OFP_cuts_LNS_100000 Gap (%) | Profit sum |
|-----|---------------------------|------------|-----------------------------|------------|
|     | Avg                | StDev           | Avg | Best | Time (s) | Avg | StDev | Avg | Best | Time (s) |
| 5_5%| 1.27                | 11.32       | 603.12 | 603.27 | 103.01 | 1.28 | 11.32 | 603.08 | 603.27 | 104.04 |
| 7_5%| 1.69                | 12.91       | 307.14 | 311.33 | 148.69 | 3.37 | 18.10 | 293.73 | 295.35 | 143.86 |

9.5.1. Exploring the convergence of the LNS heuristic

We performed additional experiments to have some insights on the convergence of the heuristic proposed. To this end, we tested stopping the execution whenever the heuristic converges to a local optimal. Precisely, instead of performing a fixed number of iterations, we halt the execution after *stalling_limit* iterations without improving — in terms of profit sum — the current best solution (see again the main algorithm in Figure 7). In this case, the PR procedure is not considered, once the heuristic stops right before the first call of PR.

As a way to also assess the influence of the initial solutions in this case, we considered the four variations of FP described in Table 9 as starting points for the LNS heuristic. The results are exposed in Table 14. Here, the columns have the same meaning as the ones of Table 11. Then, we performed the Iman-Davenport test based on the relative gaps, obtaining the statistic $F_R = 1.76$, with $p$-value = 0.15, which does not discard the null hypothesis of equal performances of all the four algorithms tested.

From Table 14, one can notice that our heuristic obtains fairly good quality solutions for the four variations tested, with extremely small average execution times. Then, the results suggest that, although the reinforcement of the initial model makes FP converge to better quality solutions
Table 14: Summary of the results obtained by the LNS heuristic when halting after $stalling\_limit$ iterations without profit sum improvement. The four FP algorithms — described in Table 9 — are tested as input for this version of the heuristic. The execution times do not consider the time spent by the FP algorithms in finding initial solutions. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and profit sums.

| Set  | Gap (%) | Profit sum | Set  | Gap (%) | Profit sum |
|------|---------|------------|------|---------|------------|
|      | Avg     | StDev      |      | Avg     | StDev      |
|      | Avg     | Best       |      | Avg     | Best       |
|      | Time (s)|            |      | Time (s)|            |
| 1_5% | 0.98    | 6.80       | 79.14 | 79.63   | 0.01       |
| 2_5% | 0.00    | 0.00       | 41.52 | 41.52   | 0.00       |
| 3_5% | 0.05    | 0.26       | 308.37| 308.50  | 0.02       |
| 4_5% | 0.70    | 1.80       | 599.85| 611.93  | 1.43       |
| 5_5% | 0.62    | 1.17       | 612.54| 617.95  | 0.14       |
| 6_5% | 0.14    | 0.32       | 373.90| 374.86  | 0.06       |
| 7_5% | 2.25    | 12.87      | 302.13| 309.70  | 0.25       |
|      | Total   | 0.74       | 5.74  | 366.27  | 370.60     | 0.30       |

For completeness, the two variations of the heuristic that performed better for the STOP instances (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) were also run to solve the original benchmark of TOP instances (Chao et al., 1996). These instances are exactly the same STOP instances described in Section 9.1, except for the lack of vertices set as mandatory. To evaluate the quality of the heuristic solutions obtained, we consider the best bounds found by the exact algorithm of Assunção and Matias (2019) (previous state-of-the-art), once the newest bounds obtained by Hanafi et al. (2020) are not reported in the publication. The results obtained are summarized in Tables 15 and 17.

In Table 15, we report, for each FP algorithm tested (FP_cuts and OFP_cuts) and instance set, four values: (i) the average and (ii) the standard deviation of the relative gaps given by $100 \cdot \frac{LB^* - LB}{LB}$. Recall that $LB^*$ is the best primal bound (for the instance) obtained by the exact algorithm of Assunção and Matias (2019), and $LB$ is the average bound (over the 10 executions) as discussed in Section 9.4, the LNS itself is able to reach near-optimal solutions regardless of the initial ones used as input.

9.6. Results for the original benchmark of TOP instances

For completeness, the two variations of the heuristic that performed better for the STOP instances (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) were also run to solve the original benchmark of TOP instances (Chao et al., 1996). These instances are exactly the same STOP instances described in Section 9.1, except for the lack of vertices set as mandatory. To evaluate the quality of the heuristic solutions obtained, we consider the best bounds found by the exact algorithm of Assunção and Matias (2019) (previous state-of-the-art), once the newest bounds obtained by Hanafi et al. (2020) are not reported in the publication. The results obtained are summarized in Tables 15 and 17.

In Table 15, we report, for each FP algorithm tested (FP_cuts and OFP_cuts) and instance set, four values: (i) the average and (ii) the standard deviation of the relative gaps given by $100 \cdot \frac{LB^* - LB}{LB}$. Recall that $LB^*$ is the best primal bound (for the instance) obtained by the exact algorithm of Assunção and Matias (2019), and $LB$ is the average bound (over the 10 executions)
Table 15: Summary of the results obtained by FP_cuts and OFP_cuts while solving the original benchmark of TOP instances. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and number of pumps.

| Set | FP_cuts | | OFP_cuts | |
|-----|---------|---------|---------|---------|
|     | Gap (%) | Profit sum | Gap (%) | Profit sum |
|     | Avg | StDev | Avg | Best | Time (s) | Avg | StDev | Avg | Best | Time (s) |
| 1   | 27.91 | 26.17 | 4.18 | 112.04 | 0.65 | 0.00 | 0.00 | 112.04 | 112.04 | 0.65 |
| 2   | 35.96 | 31.49 | 3.49 | 140.45 | 0.18 | 0.61 | 2.42 | 139.14 | 140.45 | 0.19 |
| 3   | 32.16 | 23.69 | 5.75 | 414.67 | 1.18 | 0.00 | 0.00 | 414.67 | 414.67 | 1.18 |
| 4   | 24.72 | 13.85 | 14.31 | 576.16 | 5.72 | 0.00 | 0.00 | 576.16 | 576.92 | 5.72 |
| 5   | 26.60 | 22.23 | 9.15 | 804.25 | 38.20 | -0.17 | 1.14 | 802.63 | 804.25 | 37.31 |
| 6   | 17.06 | 19.64 | 4.78 | 140.45 | 0.18 | 0.00 | 0.00 | 139.14 | 140.45 | 0.19 |
| 7   | 31.50 | 21.01 | 13.46 | 568.25 | 10.90 | 0.24 | 1.54 | 450.11 | 450.57 | 2.92 |
| Total | 27.88 | 22.88 | 8.44 | 14.18 | 26.76 | 21.68 | 29.93 | 29.47 |

Table 16: Summary of the results obtained by FP_cuts_LNS_5000 and OFP_cuts_LNS_5000 while solving the original benchmark of TOP instances. The execution times do not consider the time spent by the FP algorithms in finding initial solutions. Bold entries highlight, for each instance set, the best algorithm(s) in terms of average gaps and profit sums.

| Set | FP_cuts_LNS_5000 | | OFP_cuts_LNS_5000 | |
|-----|-----------------|---------|-----------------|---------|
|     | Gap (%) | Profit sum | Gap (%) | Profit sum |
|     | Avg | StDev | Avg | Best | Time (s) | Avg | StDev | Avg | Best | Time (s) |
| 1   | 1.67 | 9.66 | 111.07 | 112.04 | 0.65 | 0.00 | 0.00 | 112.04 | 112.04 | 0.65 |
| 2   | 6.67 | 15.14 | 131.45 | 140.45 | 0.18 | 0.61 | 2.42 | 139.14 | 140.45 | 0.19 |
| 3   | 3.17 | 13.96 | 411.22 | 414.67 | 1.18 | 0.00 | 0.00 | 414.67 | 414.67 | 1.18 |
| 4   | 0.34 | 3.11 | 801.22 | 804.23 | 38.20 | -0.17 | 1.14 | 802.63 | 804.25 | 37.31 |
| 5   | 0.38 | 2.52 | 756.16 | 756.92 | 5.72 | 0.00 | 0.00 | 756.16 | 756.92 | 5.72 |
| 6   | 0.71 | 4.63 | 449.20 | 450.57 | 3.10 | 0.24 | 1.54 | 450.11 | 450.57 | 2.92 |
| 7   | 2.66 | 12.00 | 562.26 | 568.25 | 10.90 | 0.14 | 0.27 | 567.33 | 568.55 | 10.97 |
| Total | 1.91 | 9.59 | 503.01 | 506.14 | 9.39 | 0.07 | 1.00 | 505.56 | 506.21 | 9.24 |

obtained by the corresponding FP algorithm; (iii) the average number of iterations/pumps (over the 10 executions and all instances of the set), and (iv) the average wall-clock processing time (in seconds). We highlight that, whenever $LB^* = LB = 0$, the corresponding gap of (i) is set to 0%.

In Table 16, for each LNS algorithm tested (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) and instance set, we report five values: (i) the average and (ii) the standard deviation of the relative gaps given by $100 \cdot \frac{LB^* - LB}{LB}$; (iii) the average (over the 10 executions and all instances of the set) profit sum of the solutions obtained, and (iv) the average (over all instances of the set) profit sum of the best (in each round of 10 executions) solutions found. At last, we provide (v) the average wall-clock processing time (in seconds) of the execution of the LNS steps, excluding the time spent by the FP algorithm in finding the initial solution. Also in this case, whenever $LB^* = LB = 0$, the corresponding gap of (i) is set to 0%.

In Table 17, we summarize the number of instances for which the two best variations of the
Table 17: Number of times the two best algorithms (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) reached and/or improved over the primal bounds provided by the cutting-plane algorithm of Assunção and Mateus (2019) while solving the original benchmark of TOP instances. Bold entries highlight, for each instance set, the algorithm(s) that performed best.

| Set | Reached best (#) | Improved best (#) | Reached best (#) | Improved best (#) |
|-----|------------------|-------------------|------------------|-------------------|
| FP_cuts_LNS_5000 | OFP_cuts_LNS_5000 |
| 1   | 54/54            | 0/54              | 54/54            | 0/54              |
| 2   | 33/33            | 0/33              | 33/33            | 0/33              |
| 3   | 60/60            | 0/60              | 60/60            | 0/60              |
| 4   | 55/60            | 12/60             | 53/60            | 12/60             |
| 5   | 78/78            | 2/78              | 78/78            | 3/78              |
| 6   | 42/42            | 0/42              | 42/42            | 0/42              |
| 7   | 56/60            | 1/60              | 59/60            | 1/60              |
| Total | 378/387         | 15/387            | 379/387          | 16/387            |

heuristic (FP_cuts_LNS_5000 and OFP_cuts_LNS_5000) were able to reach the primal bounds obtained by baseline exact algorithm (Assunção and Mateus, 2019). For each algorithm and instance set, we also indicate the number of cases where the LNS algorithm improved over the best solutions provided by Assunção and Mateus (2019).

10. Concluding remarks

In this work, we prove that solely finding a feasible solution for the Steiner Team Orienteering Problem (STOP) is NP-hard and propose a Large Neighborhood Search (LNS) heuristic to solve the problem. The heuristic combines classical local searches from the literature of routing problems with a memory component based on Path Relinking (PR). Our heuristic is provided with initial solutions obtained by means of the matheuristic framework known as Feasibility Pump (FP). In our implementations, FP uses as backbone a commodity-based formulation reinforced by three classes of valid inequalities, being two of them also introduced in this work.

We used the primal bounds provided by a state-of-the-art cutting-plane algorithm from the literature to guide our computational experiments, which showed the efficiency and effectiveness of the proposed heuristic in solving a benchmark of 387 instances. In particular, the heuristic could reach the best previously known bounds on 382 of these instances. Additionally, in 21 of these cases, our heuristic was even able to improve over the best known bounds. Overall, the heuristic solutions imply an average percentage gap of only 0.54% when compared to the bounds of the cutting-plane baseline.

In terms of future work directions, we first remark that the two new classes of inequalities here proposed can be naturally extended for other routing problems. In addition, we believe that our heuristic might be also successful in solving closely related orienteering problems, such as OARP and TOARP, in which arcs (instead of vertices) are sub-divided into mandatory and profitable.

Acknowledgments

This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais,
Brazil (FAPEMIG), and Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES).

References

Achterberg, T., Berthold, T., 2007. Improving the feasibility pump. Discrete Optimization 4 (1), 77 – 86.

Archetti, C., Corberán, Á., Plana, I., Sanchis, J. M., Speranza, M., 2014. The team orienteering arc routing problem. Transportation Science 48 (3), 313–463.

Archetti, C., Corberán, Á., Plana, I., Sanchis, J. M., Speranza, M. G., 2015. A matheuristic for the team orienteering arc routing problem. European Journal of Operational Research 245 (2), 392 – 401.

Archetti, C., Corberán, Á., Plana, I., Sanchis, J. M., Speranza, M. G., 2016. A branch-and-cut algorithm for the orienteering arc routing problem. Computers & Operations Research 66, 95 – 104.

Archetti, C., Hertz, A., Speranza, M. G., 2007. Metaheuristics for the team orienteering problem. Journal of Heuristics 13 (1), 49–76.

Assunção, L., Mateus, G. R., 2019. A cutting-plane algorithm for the Steiner team orienteering problem. Computers & Industrial Engineering 135, 922 – 939.

Balas, E., 1975. Facets of the knapsack polytope. Mathematical Programming 8 (1), 146–164.

Bellman, R., 1957. Dynamic Programming, 1st Edition. Princeton University Press, Princeton, NJ, USA.

Berthold, T., Lodi, A., Salvagnin, D., 2019. Ten years of feasibility pump, and counting. EURO Journal on Computational Optimization 7 (1), 1–14.

Bianchessi, N., Mansini, R., Speranza, M. G., 2018. A branch-and-cut algorithm for the team orienteering problem. International Transactions in Operational Research 25 (2), 627–635.

Bicalho, L. H., da Cunha, A. S., Lucena, A., 2016. Branch-and-cut-and-price algorithms for the degree constrained minimum spanning tree problem. Computational Optimization and Applications 63 (3), 755–792.

Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. 4OR 5 (3), 211–230.

Chao, I.-M., Golden, B. L., Wasil, E. A., 1996. The team orienteering problem. European Journal of Operational Research 88 (3), 464–474.

Cormen, T. H., Stein, C., Rivest, R. L., Leiserson, C. E., 2001. Introduction to Algorithms, 2nd Edition. McGraw-Hill Higher Education.

Dang, D.-C., El-Hajj, R., Moukrim, A., 2013a. A branch-and-cut algorithm for solving the team orienteering problem. In: Gomes, C., Sellmann, M. (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Vol. 7874 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 332–339.
Dang, D.-C., Guibadj, R. N., Moukrim, A., 2013b. An effective PSO-inspired algorithm for the team orienteering problem. European Journal of Operational Research 229 (2), 332–344.

Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30.

Dezsö, B., Jüttner, A., Péter Kovács, 2011. LEMON — an open source C++ graph template library. Electronic Notes in Theoretical Computer Science 264 (5), 23 – 45, proceedings of the Second Workshop on Generative Technologies (WGT) 2010.

El-Hajj, R., Dang, D.-C., Moukrim, A., 2016. Solving the team orienteering problem with cutting planes. Computers & Operations Research 74, 21 – 30.

Fischetti, M., Glover, F., Lodi, A., 2005. The feasibility pump. Mathematical Programming 104 (1), 91–104.

Friedman, M., 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32 (200), 675–701.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

Glover, F., 1997. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. Springer US, Boston, MA, pp. 1–75.

Goldberg, A. V., Tarjan, R. E., 1988. A new approach to the maximum-flow problem. J. ACM 35 (4), 921–940.

Gu, Z., Nemhauser, G. L., Savelsbergh, M. W. P., 1998. Lifted cover inequalities for 0–1 integer programs: Computation. INFORMS Journal on Computing 10 (4), 427–437.

Gunawan, A., Lau, H. C., Vansteenwegen, P., 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research 255 (2), 315 – 332.

Hanafi, S., Mansini, R., Zanotti, R., 2020. The multi-visit team orienteering problem with precedence constraints. European Journal of Operational Research 282 (2), 515–529.

Iman, R., Davenport, J., 1980. Approximations of the critical region of the Friedman statistic. Communications in Statistics 9, 571–595.

Kaparis, K., Letchford, A. N., 2008. Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 186 (1), 91 – 103.

Ke, L., Archetti, C., Feng, Z., 2008. Ants can solve the team orienteering problem. Computers & Industrial Engineering 54 (3), 648–665.

Ke, L., Yang, W., 2017. Reformulation and metaheuristic for the team orienteering arc routing problem. In: Tan, Y., Takagi, H., Shi, Y., Niu, B. (Eds.), Advances in Swarm Intelligence. ICSI 2017. Lecture Notes in Computer Science. Vol. 10386. Springer International Publishing, Cham, pp. 494–501.

Ke, L., Zhai, L., Li, J., Chan, F. T., 2016. Pareto mimic algorithm: An approach to the team orienteering problem. Omega 61, 155 – 166.
Keshtkaran, M., Ziarati, K., Bettinelli, A., Vigo, D., 2016. Enhanced exact solution methods for the team orienteering problem. International Journal of Production Research 54 (2), 591–601.

Kim, B.-I., Li, H., Johnson, A. L., 2013. An augmented large neighborhood search method for solving the team orienteering problem. Expert Systems with Applications 40 (8), 3065–3072.

Letchford, A. N., Nasiri, S. D., Theis, D. O., 2013. Compact formulations of the Steiner traveling salesman problem and related problems. European Journal of Operational Research 228 (1), 83–92.

Lin, S., 1965. Computer solutions of the traveling salesman problem. The Bell System Technical Journal 44 (10), 2245–2269.

Lin, S.-W., 2013. Solving the team orienteering problem using effective multi-start simulated annealing. Applied Soft Computing 13 (2), 1064–1073.

Nemenyi, P., 1963. Distribution-free Multiple Comparisons. Princeton University.

Poggi, M., Viana, H., Uchoa, E., 2010. The team orienteering problem: Formulations and branch-cut and price. In: ATMOS 2010 - 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Liverpool, United Kingdom, September 6-10, 2010. pp. 142–155.

Riera-Ledesma, J., Salazar-González, J. J., 2017. Solving the team orienteering arc routing problem with a column generation approach. European Journal of Operational Research 262 (1), 14 – 27.

Salazar-Aguilar, M. A., Langevin, A., Laporte, G., 2014. The multi-district team orienteering problem. Computers & Operations Research 41, 76 – 82.

Samer, P., Urrutia, S., 2015. A branch and cut algorithm for minimum spanning trees under conflict constraints. Optimization Letters 9 (1), 41–55.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.-F. (Eds.), Principles and Practice of Constraint Programming — CP98. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 417–431.

Sheskin, D. J., 2007. Handbook of Parametric and Nonparametric Statistical Procedures, 4th Edition. Chapman & Hall/CRC.

Souffriau, W., Vansteenwegen, P., Berghe, G. V., Oudheusden, D. V., 2010. A path relinking approach for the team orienteering problem. Computers & Operations Research 37 (11), 1853–1859.

Tang, H., Miller-Hooks, E., 2005. A TABU search heuristic for the team orienteering problem. Computers & Operations Research 32 (6), 1379–1407.

Tomita, E., Tanaka, A., Takahashi, H., 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science 363 (1), 28 – 42.

Tricoire, F., Romauch, M., Doerner, K. F., Hartl, R. F., 2010. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37 (2), 351 – 367.
Vansteenwegen, P., Souffriau, W., Berghe, G. V., Oudheusden, D. V., 2009. A guided local search metaheuristic for the team orienteering problem. European Journal of Operational Research 196 (1), 118–127.

Vansteenwegen, P., Souffriau, W., Oudheusden, D. V., 2011. The orienteering problem: A survey. European Journal of Operational Research 209 (1), 1 – 10.

Vidal, T., Maculan, N., Penna, P., Ochi, L., 2015. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. Transportation Science 50 (2), 363–761.

Wesselmann, F., Suhl, U. H., 2012. Implementing cutting plane management and selection techniques. Tech. rep., University of Paderborn, Paderborn, Germany.

Wolsey, L. A., 1975. Faces for a linear inequality in 0–1 variables. Mathematical Programming 8 (1), 165–178.