INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2022) 1–35
DOI: 10.1111/itor.13160

The bus sightseeing problem

Qian Hu[a], Zhenzhen Zhang[b], Roberto Baldacci[c,*], Christos D. Tarantilis[d] and Emmanouil Zachariadis[d]

[a]*School of Management and Engineering, Nanjing University, Nanjing 210093, China*
[b]*School of Economics and Management, Tongji University, Shanghai 200092, China*
[c]*Division of Engineering Management and Decision Sciences (EMDS), College of Science and Engineering (CSE), Hamad Bin Khalifa University (HBKU), Doha 5825, Qatar*
[d]*Department of Management Science & Technology, Athens University of Economics & Business, 76 Patission Street, Athens 10434, Greece*
E-mail: huqian@nju.edu.cn [Hu]; zhenzhenzhang222@gmail.com [Zhang]; rbaldacci@hbku.edu.qa [Baldacci]; tarantil@aueb.gr [Tarantilis]; ezach@aueb.gr [Zachariadis]

Received 21 December 2021; received in revised form 30 April 2022; accepted 17 May 2022

Abstract

The basic characteristic of vehicle routing problems with profits (VRPP) is that locations to be visited are not predetermined. On the contrary, they are selected in pursuit of maximizing the profit collected from them. Significant research focus has been directed toward profitable routing variants due to the practical importance of their applications and their interesting structure, which jointly optimizes node selection and routing decisions. Profitable routing applications arise in the tourism industry aiming to maximize the profit score of attractions visited within a limited time period. In this paper, a new VRPP is introduced, referred to as the bus sightseeing problem (BSP). The BSP calls for determining bus tours for transporting different groups of tourists with different preferences on touristic attractions. Two interconnected decision levels have to be jointly tackled: assignment of tourists to buses and routing of buses to the various attractions. A mixed-integer programming formulation for the BSP is provided and solved by a Benders decomposition algorithm. For large-scale instances, an iterated local search based metaheuristic algorithm is developed with some tailored neighborhood operators. The proposed methods are tested on a large family of test instances, and the obtained computational results demonstrate the effectiveness of the proposed solution approaches.

Keywords: branch and cut; local search based metaheuristic; orienteering; profit collection; sightseeing; vehicle routing

*Corresponding author.

Qian Hu and Zhenzhen Zhang are co-first authors.

1. Introduction

The typical challenge for a traveler visiting a touristic destination for a limited period of time is to design a schedule that takes full advantage of the various points of interest (POIs), such as museums, monuments, art galleries, restaurants, etc. Generally, a tourist can be interested in planning his/her sightseeing activities strictly based on his/her personal preferences and on the amount of time and budget available. This problem has been introduced in the literature as the tourist trip design problem (TTDP). Under the TTDP model, a tourist defines his/her individual preferences and interests on specific POIs. The solution is a multiperiod set of *personalized* tours that maximize the predetermined tourist preference levels. The TTDP can be regarded as a variant of the basic orienteering problem (OP) (Vansteenwegen et al., 2019b; Ruiz-Meza and Montoya-Torres, 2022).

Alternatively, a tourist can be interested in tours or packages offered by sightseeing or travel companies. The companies generally operate with fixed bus fleets, which are dispatched to a subset of POIs according to the preferences of the tourists on board, under various operational constraints. Motivated from the perspective of these companies, other than that of a tourist as in the TTDP, we introduce the bus sightseeing problem (BSP). Generally speaking, the BSP considers a set of tourist groups and a set of POIs, and each tourist group is composed of several people with specific preferences on the POIs. The objective is to (i) assign tourist groups to buses and (ii) determine the bus tours (POI subset determination and POI visit ordering), to maximize the tourist preferences, while taking into account the operational constraints, that is, bus capacity and opening hours. Figure 1 depicts an example of a BSP solution that maximizes the total preference score over all tourist groups.

In the BSP, two interconnected decision levels have to be jointly tackled: assignment of tourists to buses and routing of buses to various attractions. This is different from the classical TTDP and vehicle routing problems with profits (VRPP), in which the profits are associated with locations and the total collected profit is computed by the sum of the profits of the visited locations. With respect to the existing literature on VRPP (Section 2), the BSP exhibits a more generalized profit collecting

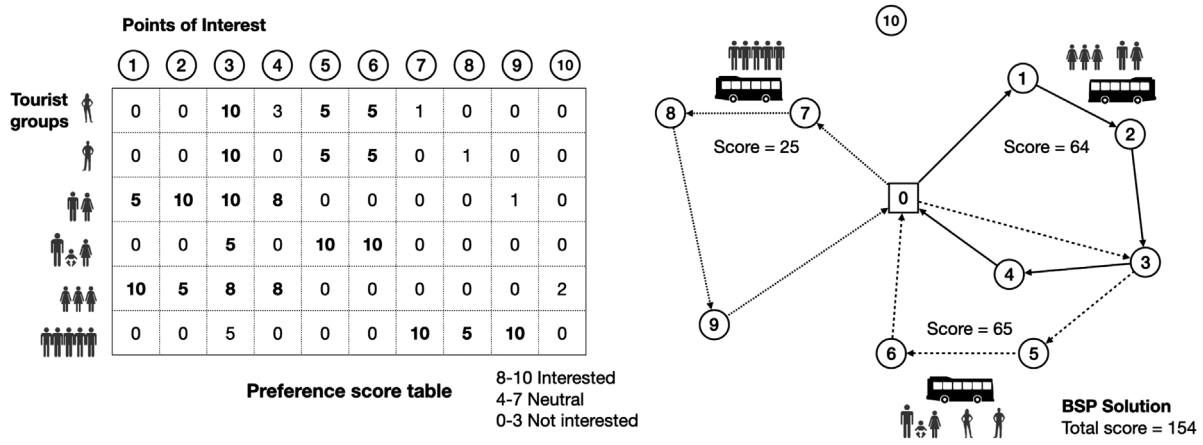

Fig 1. A BSP example that maximizes tourist preferences.

structure. By formally introducing the BSP (Section 3), we show that the BSP generalizes the OP with time windows (OPTW) and the team OPTW (TOPTW).

The main aim of this paper is to investigate bounding techniques and an exact method for the BSP. We describe a mathematical formulation for the BSP (Section 4). Based on the formulation, we derive valid dual bounds by solving the linear programming (LP) relaxation of the formulation. The dual bounds are further strengthened by a set of valid inequalities. A branch-and-cut (B&C) approach, which uses the valid inequalities and Benders decomposition techniques, is developed to solve the problem to optimality (Section 5). Furthermore, an iterated local search (ILS) based metaheuristic algorithm is developed to efficiently compute primal bounds (Section 6). A key feature of the proposed ILS algorithm is the use of two mixed-integer programming (MIP) based moves to search neighborhoods. Finally, we perform extensive computational experiments on a set of 576 newly constructed benchmark instances to assess the quality of the proposed solution methods (Section 7). The computational results show that our exact method can optimally solve test instances with up to 20 POIs and 30 tourist groups. Moreover, the results show that the ILS produces improved solutions for the instances not solved to optimality by the exact method, and can effectively compute solutions for larger-scale instances.

2. Related work

The BSP belongs to the class of VRPP (Archetti et al., 2014; Vansteenwegen et al., 2019b) where, in general, a profit is associated with each location to indicate its attractiveness. However, any BSP route is associated with a cost, as well as a profit value. Another central feature of the VRPP models, which distinguishes them from the basic family of routing models, is that location service is optional and the decision maker is responsible for identifying the location subset to be visited. Variants of VRPP ranging from single to multivehicle models inspired by practical transportation applications can be found in the survey paper of Feillet et al. (2005), the more recent review of Archetti et al. (2014), and the recent book of Vansteenwegen et al. (2019b). To the best of our knowledge, the BSP model has never been studied before. TOPTW is one of the most closely related problems to BSP. Thus, in the following we focus on methods proposed for the TOPTW and some interesting TOPTW applications in the tourism industry.

To the best of our knowledge, no exact approaches have been proposed for solving the TOPTW. Regarding its special case with a single vehicle (OPTW), an exact algorithm based on bidirectional dynamic programming is proposed in Righini and Salani (2006). On the other hand, a number of heuristic techniques have been proposed for both TOPTW and OPTW. A guided local search (LS) and an ILS metaheuristic were proposed for the TOPTW in Vansteenwegen et al. (2009). Later, many other metaheuristics have been proposed, such as the hybridized greedy randomized adaptive search procedure (GRASP) with the evolutionary LS (Labadie et al., 2011), the LP-based variable neighborhood search (VNS) (Labadie et al., 2012), the iterative framework based on LS procedure and simulated annealing (SA) (Hu and Lim, 2014), the iterated LS (ILS), and a hybridization of SA and ILS (Gunawan et al., 2017), to name a few. The reader is referred to Vansteenwegen et al. (2019b) for a comprehensive review of the benchmark instances and the state-of-the-art solution techniques of the OPTW and the TOPTW.

We next list some interesting TOPTW applications in the tourism industry.

Souffriau et al. (2013) introduced the multiconstraint team OP with multiple time windows (MC-TOP-MTW), where a set of POIs is given, each with a service time, one or more time windows (TW), and a profit score. The goal is to maximize the sum of the profits collected by a fixed number of tours. The generated tours are subject to various operational requirements, such as maximum tour duration constraints and hard TW of POIs. The authors provide a mathematical formulation for the MC-TOP-MTW, and an ILS equipped with a GRASP. The algorithm was tested on instances involving from 48 to 288 POIs and 1–4 tours. Garcia et al. (2013) introduced the tourist planning problem that integrates public transportation, as the time-dependent TOPTW (TD-TOPTW). The problem calls for determining personalized tourist routes in real time. The authors developed and compared two different heuristic approaches for the TD-TOPTW and reported results for actual data taken from the city of San Sebastian. Malucelli et al. (2015) studied a problem calling for the optimal design of cycle tourist itineraries. It assumes that the profit of nodes or edges can be collected several times but with a decreased rate. An integer programming formulation of the problem based on the OP and the multicommodity network design problem models is described. For a comprehensive overview, the interested reader is referred to the paper of Gavalas et al. (2014) and to Vansteenwegen et al. (2019b).

3. Problem description

In this section, we formally introduce the BSP and discuss its complexity by showing that the OPTW and the TOPTPW are special cases of the BSP.

The BSP is defined on a complete digraph $G = (V, A)$ where V is the vertex set and A is the set of arcs. The vertex set is composed of vertex 0 and a vertex set P representing the various POIs of the examined touristic destination, that is, $V = \{0\} \cup P$. Vertex 0 represents a central location, where vehicle or bus routes originate and terminate. Hereafter, we use the terms vehicle or bus interchangeably.

Each vertex $i \in P$ is associated with a hard TW $[e_i, l_i]$, which defines the earliest and latest time for the start of a visit at POI i. In addition, a visit time v_i is also considered, representing the time necessary for a POI i visit. Note that if a bus arrives at i before e_i, tourists must wait until the opening time e_i of this particular POI. In real scenarios, POIs have early opening times so waiting times at POIs are generally not an issue. Each arc $(i, l) \in A$, $i, l \in V$, $i \neq l$, is associated with a travel time $\hat{t}_{il}$. The travel time $\hat{t}_{il}$ can be modified to include the visit time at vertex i, thus obtaining a modified travel time t_{il}, that is, $t_{il} = v_i + \hat{t}_{il}$. We assume that matrix t_{il} satisfies the triangle inequality. At the central location 0, a set K of buses is available. With each bus $k \in K$ is associated a capacity Q_k representing the maximum number of people on board. Moreover, vertex 0 is also associated with a TW $[e_0, l_0]$, representing a common working time period for all buses in K.

The BSP also considers a set of *tourist groups* T. Each group $j \in T$ consists of q_j people. In addition, each group declares how important it is for this group to visit a particular POI in P. More precisely, with each group $j \in T$ and vertex $i \in P$ pair, it is associated a nonnegative profit p_{ji}, which conveys the satisfaction level (hereafter referred to as profit) enjoyed by j by visiting POI i. If a group $j \in T$ is assigned to a bus $k \in K$, which visits vertex $i \in P$, then the corresponding profit p_{ji} is enjoyed (collected) by group j.

The aim of the BSP is to jointly decide the assignment of groups to buses, and for each of these buses, design the corresponding route. More precisely, the BSP calls for the *group-vehicle assignment* and the *routing of vehicles*, which maximizes the total profit enjoyed by all groups. The two decision levels are subject to the following constraints:

(A) *Group-vehicle assignment*.
  (i) Each tourist group must be assigned to exactly one vehicle.
  (ii) The total number of people assigned to a vehicle must not exceed the capacity of the vehicle.
(B) *Routing of vehicles*.
  (i) Each vehicle performs at most one route.
  (ii) Each vehicle route starts from vertex 0, and terminates at vertex 0.
  (iii) Each vehicle visits any POI vertex at most once, whereas a POI vertex may be visited at most L times with $L \leq |K|$. Since a POI can be visited several times by different vehicles, parameter L sets an upper limit on the number of vehicles that can visit a POI.
  (iv) Each vehicle may arrive at a POI within the corresponding TW. If it arrives earlier, it must wait for the TW opening.

The link between the OPTW/TOPTW and the BSP can be summarized as follows. The OPTW is defined on a complete digraph $G' = (V', A')$, where V' is the vertex set and A' is the arc set. Vertex 0 represents the depot. Let $\overline{p}_i$ denote the nonnegative profit associated with $i \in V'$ (with $\overline{p}_0 = 0$), and let $\overline{t}_{il}$ be the nonnegative travel time associated with arc $(i, l) \in A'$ that includes the visit time at vertex i. In addition, let $[\overline{e}_i, \overline{l}_i]$ be the TW associated with vertex $i \in V'$. The OPTW calls for the determination of a single route that maximizes the total profit collected, by visiting each vertex in $V' \setminus \{0\}$ at most once and within its TW.

Any OPTW instance can be converted into an equivalent BSP instance as follows:

(i) Define graph $G = (V, A)$ by setting $V = V'$, $A = A'$, $P = V' \setminus \{0\}$ and $t_{il} = \overline{t}_{il}$, $\forall (i, l) \in A$.
(ii) Define $|K| = 1$, $Q_0 = 1$ and $L = 1$.
(iii) Define $|T| = 1$, $q_0 = 1$ and $p_{0i} = \overline{p}_i$, $\forall i \in V' \setminus \{0\}$.
(iv) Define $[e_i, l_i] = [\overline{e}_i, \overline{l}_i]$, $\forall i \in V$.

The TOPTW generalizes the OPTW by calling for the determination of m routes maximizing the total profit collected. Any TOPTW instance can be converted into an equivalent BSP instance by generating graph G and the TW as in points (i) and (iv), and the following additional definitions:

(i) $|K| = m$, $Q_k = 1$, $\forall k \in K$, and $L = 1$; and
(ii) $|T| = m$, $q_j = 1$, $\forall j \in T$, and $p_{ji} = \overline{p}_i$, $\forall j \in T$, $\forall i \in V' \setminus \{0\}$.

Because the OPTW is $\mathcal{NP}$-hard, so is the BSP.

3.1. Link between the TTDP and the BSP

In the literature, the term TTDP is used to refer to a general class of problems associated with the construction of personalized tourist itineraries and no specific problem definition is reported.

Indeed, as reported by Vansteenwegen et al. (2019a) and Ruiz-Meza and Montoya-Torres (2022), almost all papers in the TTDP literature use the OP (and other extended variants) to formulate the TTDP. Moreover, various extensions of the TTDP can also be formulated as the variants of the TOP, OPTW, or TOPTW. Vansteenwegen et al. (2019a) listed several TTDP variants and Ruiz-Meza and Montoya-Torres (2022) gave a classification of different variants and solution techniques. Variants include heterogeneous preferences of tourists, multiple TW, different classes of users, multiobjective function, to name a few. The reader is referred to Vansteenwegen et al. (2019a) and Ruiz-Meza and Montoya-Torres (2022) for a comprehensive list of the different variants.

As shown above, as an application of the TTDP, our problem generalizes the TOPTW by introducing the following two main characteristics:

(i) the vehicle fleet is assumed to be heterogeneous and, most importantly,
(ii) the profits are heterogeneous, that is, under the BSP model, the total profit enjoyed depends on the group-vertex assignments, whereas for the classical OP/TOP, the total profit depends only on the vertices visited.

In the literature, TTDP are generally formulated based on OP/TOP/TOPTW mathematical formulations. However, due to the structure of heterogeneous profits of the BSP, existing OP/TOP/TOPTW formulations cannot be used directly to solve the problem, and in the next section we introduce a mathematical formulation for the BSP where additional decision variables are introduced to model the heterogeneous profits.

From the methodological point of view, to the best of our knowledge, no exact approaches have been proposed for solving the TOPTW, and existing exact methods for the OP/TOP cannot be easily extended the BSP due to the above two main characteristics. In this paper, we therefore describe a new exact method for the problem (see Section 5) and a metaheuristic (see Section 6). More specifically, the metaheuristic uses an ILS solution framework, widely adopted to effectively solve TTDP variants. In addition, our heuristic algorithm also relies on the solution of OPTW and TOPTW instances in the context of MIP-based moves used by the LS algorithm.

4. Mathematical formulation and valid inequalities

This section provides an MIP formulation for the BSP. It also describes properties of the proposed formulation, together with valid inequalities that can be used to strengthen the dual bounds derived from the LP-relaxation of the formulation.

4.1. Formulation

A bus route or simply *route* $R = (i_0 = 0, i_1, i_2, \ldots, i_{n-1}, i_n = 0)$ is defined as a simple circuit in G. It starts at the bus station 0 at time e_0 and ends at the station no later than l_0. In between, it visits the POI subset represented by vertices $V(R) = \{i_1, \ldots, i_{n-1}\}$. Route R is *feasible* if the visit start time $\tau_{i_h}, \forall i_h \in V(R) \cup \{i_n\}$, satisfies $\max\{\tau_{i_{h-1}} + t_{i_{h-1}i_h}, e_{i_h}\} \leq \tau_{i_h} \leq l_{i_h}$, where i_{h-1} is the vertex visited immediately before i_h in route R (and $\tau_0 = e_0$).

The proposed BSP formulation uses the following decision variables:

- y_{ji}^k: binary variables, equal to 1, if and only if vertex $i \in P$ is visited by group $j \in T$ using vehicle $k \in K$;
- z_j^k: binary variables, equal to 1, if and only if group $j \in T$ is assigned to vehicle $k \in K$;
- s_i^k: binary variables, equal to 1, if and only if vertex $i \in V$ is visited by vehicle $k \in K$;
- x_{il}^k: binary variables, equal to 1 if and only if arc $(i, l) \in A$ is traversed by vehicle $k \in K$;
- δ_i^k: nonnegative continuous variables representing the start time at vertex $i \in V$ visited by vehicle $k \in K$; if vertex i is not visited by vehicle k, we have $\delta_i^k = 0$.

We split the BSP constraints into two distinct blocks. The first block handles the assignment of tourist groups to buses and the POI subset determination (which POIs are going to be visited) for each vehicle, whereas the second block addresses the routing of buses. Let X_1 denote a set of vectors $\mathbf{y} \in \{0, 1\}^{|K|\times|T|\times|P|}$, $\mathbf{z} \in \{0, 1\}^{|K|\times|T|}$ and $\mathbf{s} \in \{0, 1\}^{|K|\times|V|}$, satisfying the following constraints (first constraint block):

$$\sum_{k\in K} s_i^k \le L, \qquad \forall i \in P, \tag{1a}$$

$$s_0^k \ge s_i^k, \qquad \forall i \in P, k \in K, \tag{1b}$$

$$y_{ji}^k \le s_i^k \le \sum_{h\in T} y_{hi}^k, \qquad \forall i \in P, j \in T, k \in K, \tag{1c}$$

$$y_{ji}^k \le z_j^k \le \sum_{l\in P} y_{jl}^k, \qquad \forall i \in P, j \in T, \forall k \in K, \tag{1d}$$

$$y_{ji}^k \ge s_i^k + z_j^k - 1, \qquad \forall i \in P, j \in T, k \in K, \tag{1e}$$

$$\sum_{j\in T} q_j z_j^k \le Q_k, \qquad \forall k \in K, \tag{1f}$$

$$\sum_{k\in K} z_j^k = 1, \qquad \forall j \in T. \tag{1g}$$

Constraints (1a) state that a vertex $i \in P$ can be visited at most L times by the vehicles whereas constraint (1b) impose that $s_0^k = 1$ if at least one POI is visited by vehicle k. Constraints (1c)–(1e) are linking constraints among the different set of variables. Constraints (1f) guarantee that the vehicle capacities are not violated. Finally, constraints (1g) force each group j to be assigned to exactly one vehicle.

Set X_2 is defined as the set of vectors $\mathbf{s} \in \{0, 1\}^{|K|\times|V|}$, $\mathbf{x} \in \{0, 1\}^{|K|\times|A|}$, and $\delta \in \mathbb{R}_+^{|K|\times|V|}$ satisfying the following constraints (second constraint block):

$$\sum_{l\in\Gamma_i^+} x_{il}^k = s_i^k, \qquad \forall i \in V, k \in K, \tag{2a}$$

$$\sum_{l \in \Gamma_i^-} x_{li}^k = s_i^k, \qquad \forall i \in V, k \in K, \tag{2b}$$

$$\delta_i^k + t_{il} - (1 - x_{il}^k)M \leq \delta_l^k, \qquad \forall (i,l) \in A, k \in K, \tag{2c}$$

$$e_i s_i^k \leq \delta_i^k \leq l_i s_i^k, \qquad \forall i \in V, k \in K, \tag{2d}$$

where $\Gamma_i^+ = \{h \in V : (i,h) \in A\}$ and $\Gamma_i^- = \{h \in V : (h,i) \in A\}$ are the sets of *successors* and *predecessors* of vertex $i \in V$ in G, respectively, and M is a sufficiently large constant, for example, $M = \sum_{(i,l)\in A} t_{il}$. For a given vertex i and vehicle k such that $s_i^k = 1$, the out-degree (2a) and in-degree (2b) constraints ensure that there will be exactly one arc entering and exactly one arc leaving vertex i. Constraints (2c) and (2d) guarantee that the bus routes respect all TW of visited vertices (POIs and bus station). From a different viewpoint, this second constraint block ensures that feasible routes exist for a given vector $\mathbf{s} \in \{0,1\}^{|K|\times|V|}$.

Based on the above definitions, the BSP is formulated as the following MIP model:

$$(F) \qquad z(F) = \max \sum_{k \in K} \sum_{j \in T} \sum_{i \in P} p_{ji} y_{ji}^k \tag{3a}$$

$$\text{s.t. } (\mathbf{y}, \mathbf{z}, \mathbf{s}) \in X_1, \tag{3b}$$

$$(\mathbf{s}, \mathbf{x}, \delta) \in X_2. \tag{3c}$$

The objective function (3a) maximizes the total profit enjoyed by all tourist groups. Constraints (3b) and (3c) represent the two blocks of constraints described earlier.

Let LF denote the LP-relaxation of formulation F and let $z(LF)$ represent its optimal solution cost. The following property holds about F and LF.

Proposition 1. *In any optimal LF solution* $(\mathbf{y}, \mathbf{z}, \mathbf{s}, \mathbf{x}, \delta)$ *we have*

$$y_{ji}^k = \min\{s_i^k, z_j^k\}, \ \forall i \in P, j \in T, k \in K. \tag{4}$$

Proof. From constraints (1c) to (1e) we have

$$\max\{s_i^k + z_j^k - 1, 0\} \leq y_{ji}^k \leq \min\{s_i^k, z_j^k\}, \ \forall i \in P, j \in T, k \in K. \tag{5}$$

Hence, due to the maximization objective of LF, equations (4) hold. □

The above property indicates that imposing integrality on variables $\mathbf{s}$ and $\mathbf{z}$ suffices to guarantee integrality of variables $\mathbf{y}$. Thus, variables $\mathbf{y}$ can be relaxed to continuous variables.

4.2. Improving relaxation LF

In this section, we describe valid inequalities that can be used to improve the quality of the dual bounds obtained from formulation LF. The corresponding separation procedures are then described in Section 5.1.

4.2.1. Capacity constraints

The following valid inequalities for LF ensure that the capacity of a bus cannot be violated at any vertex visited by the bus.

Proposition 2. *The following inequalities are valid for LF:*

$$\sum_{j \in T} q_j y_{ji}^k \leq Q_k s_i^k, \ \forall i \in P, k \in K. \tag{6}$$

Proof. We have the following two cases:

(i) If $s_i^k = 1$, then $\sum_{j \in T} q_j y_{ji}^k \leq \sum_{j \in T} q_j z_j^k \leq Q_k s_i^k = Q_k$.
(ii) If $s_i^k = 0$, then $\sum_{j \in T} q_j y_{ji}^k = 0 \leq Q_k s_i^k$. □

4.2.2. Inequalities from the associated knapsack polytope

We consider cover and lifted cover inequalities based on knapsack constraints (1f) (Crowder et al., 1983; Nemhauser and Wolsey, 1988). For a given $k \in K$, let the subset of tourist groups $C \subseteq T$ be a *cover* such that $\sum_{j \in C} q_j > Q_k$ and let $\mathcal{C}^k$ be the set of all covers for vehicle k. The following cover inequalities are valid for LF:

$$\sum_{j \in C} z_j^k \leq |C| - 1, \quad \forall C \in \mathcal{C}^k, k \in K. \tag{7}$$

Cover inequalities can be lifted by letting $q^{\max} = \max_{j \in C}\{q_j\}$ and $\hat{T} = \{j \in T \setminus C : q_j \geq q^{\max}\}$. Then, the following lifted cover inequality is valid for LF:

$$\sum_{j \in C \cup \hat{T}} z_j^k \leq |C| - 1, \quad \forall C \in \mathcal{C}^k, k \in K. \tag{8}$$

4.2.3. Infeasible path elimination constraints

These constraints filter out POI vertex sets that cannot be feasibly visited by a route due to the TW constraints imposed by the BSP. Let $D \subseteq P$ denote an infeasible set of vertices such that there does not exist any feasible route containing all D vertices and respecting the TW constraints. Let $\mathcal{D}$ include all such infeasible sets. The following inequalities are valid for LF:

$$\sum_{i \in D} s_i^k \leq |D| - 1, \quad \forall D \in \mathcal{D}, k \in K. \tag{9}$$

These constraints can also be expressed in terms of the **y** variables, as shown in the following proposition.

Proposition 3. *The following inequalities are valid for LF:*

$$\sum_{i \in D} \sum_{k \in K} y_{ji}^k \leq |D| - 1, \ \forall D \in \mathcal{D}, j \in T. \tag{10}$$

Proof. Let $w_{ji} = \sum_{k \in K} y^k_{ji}$. Obviously, $w_{ji} \in \{0, 1\}$ in any feasible F solution, i.e., $w_{ji} = 1$ if group j visits vertex i, and $w_{ji} = 0$ otherwise. Since D represents an infeasible set of vertices, then $\sum_{i \in D} w_{ji} \leq |D| - 1$ is a valid inequality, and therefore (10) hold. □

4.2.4. Symmetry breaking constraints

We also use a set of symmetry breaking inequalities, to avoid equivalent solutions obtained by interchanging index k. These are useful for instances involving buses of equal capacity, that is, $Q_{k_1} = Q_{k_2}$, $\forall k_1, k_2 \in K, k_1 \neq k_2$. Let σ be a permutation of the group set T. We can assume that $h(1) \leq h(2) \leq \cdots \leq h(|K|)$, where $h(k) = \min\{j \in T : z^k_{\sigma(j)} = 1\}$, $k = 1, \ldots, |K|$ is the smallest group number (w.r.t. permutation σ) of the groups visited by vehicle k ($h(k) = \infty$ if vehicle k is not used). Then the following constraints hold:

$$\begin{aligned} & z^1_{\sigma(1)} = 1, \\ & z^k_{\sigma(i)} \leq \sum_{j \in \{1,\ldots,i-1\}} z^{k-1}_{\sigma(j)}, \quad \text{for all } k \geq 3, k \in K, \text{ and } i \geq 2, i \in T. \end{aligned} \tag{11}$$

In this way, vehicles with smaller indices are assigned to groups with smaller indices.

5. An exact method for the BSP

MIP formulations such as formulation F are notoriously very hard to solve by means of general purpose MIP solvers based on B&C approaches. Nevertheless, speedup of some orders of magnitude with respect to the best MIP solvers on the market can be achieved using logic-based solution methods (Hooker, 2000; Codato and Fischetti, 2006; Zhang et al., 2021). In this section, we describe an exact BSP solution method based on the Benders decomposition, formally developed by Benders (1962) to efficiently solve MIPs. The Benders decomposition partitions the original problem into two problems, that is, an integer master problem (MP) and a linear slave problem or subproblem (SP). The MP is solved for a decision variable subset, whereas the rest of the values are decided by solving the SP. If the SP is found to be infeasible, new cuts are incorporated in the MP, which is re-solved in a cyclic framework. Geoffrion (1972) has extended the Benders decomposition to a larger class of mathematical programming problems. Finally, the works of Hooker (2000) and Hooker and Ottosson (2003) further extend the Benders strategy by considering more general subproblems for generating MP cuts. This approach is referred to as a logic-based Benders decomposition. Later, it was specialized to MIP by Codato and Fischetti (2006) who introduced the so-called combinatorial Benders cuts.

Following a logic-based Benders approach, the master problem MP can be defined as follows:

$$(MP) \qquad z(MP) = \max \sum_{k \in K} \sum_{j \in T} \sum_{i \in P} p_{ji} y^k_{ji} \tag{12a}$$

$$\text{s.t. } (\mathbf{y}, \mathbf{z}, \mathbf{s}) \in X_1. \tag{12b}$$

Algorithm 1. Benders decomposition method

Set $UB = \infty$ and $LB = -\infty$; // Initialization
while $UB > LB$ **do**
  Solve problem MP ;
  if MP *is infeasible* **then**
    return no feasible BSP solution found;
  else
    Let $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$ be the optimal MP solution of cost $\overline{z}$;
    Set $UB = \overline{z}$;
    Solve subproblem SP with $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$;
    if SP *is infeasible* **then**
      Add the infeasibility cut (13) to problem MP;
    else
      SP admits a feasible and integer solution $(\overline{\mathbf{x}}, \overline{\boldsymbol{\delta}})$;
      Let $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}}, \overline{\mathbf{x}}, \overline{\boldsymbol{\delta}})$ be the corresponding BSP feasible solution;
      Set $LB = \overline{z}$;
      return BSP optimal solution $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}}, \overline{\mathbf{x}}, \overline{\boldsymbol{\delta}})$;
    end
  end
end

Problem MP is $\mathcal{NP}$-hard since it reduces to the generalized assignment problem (GAP) (Martello and Toth, 1990) when variables $\mathbf{s}$ are fixed. Given a solution $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$ of MP, problem SP is a *feasibility* or *separation* subproblem calling for the evaluation of $(\overline{\mathbf{s}}, \mathbf{x}, \delta)$ such that $(\overline{\mathbf{s}}, \mathbf{x}, \delta) \in X_2$.

For a given MP solution $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$, there are two possible outcomes w.r.t. the associated subproblem SP:

(i) Problem SP is infeasible for $\overline{s}$. The following *infeasibility* cut is added to MP:

$$\sum_{k \in K} \sum_{i \in P: \overline{s}_i^k = 0} s_i^k + \sum_{k \in K} \sum_{i \in P: \overline{s}_i^k = 1} (1 - s_i^k) \geq 1, \tag{13}$$

to render solution $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$ infeasible. Indeed, if $s = \overline{s}$ the left-hand side of inequality (13) is equal to 0, hence the inequality cuts off the point $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}})$ from the master MP.

(ii) Problem SP is feasible for $\overline{s}$. Let $(\overline{\mathbf{x}}, \overline{\delta})$ be the solution of SP. Then, solution $(\overline{\mathbf{y}}, \overline{\mathbf{z}}, \overline{\mathbf{s}}, \overline{\mathbf{x}}, \overline{\delta})$ is an optimal BSP solution of total profit equal to $\sum_{k \in K} \sum_{j \in T} \sum_{i \in P} p_{ji} \overline{y}_{ji}^k$.

The proposed solution algorithm is summarized by Algorithm 1.

Over the course of the algorithm, UB corresponds to a valid dual bound on $z(F)$ whereas at the end of the algorithm $LB = z(F)$ (we assume $z(F) = -\infty$ if there is no feasible solution).

Note that the proposed algorithm terminates with an optimal solution whenever a feasible SP solution is found. In addition, if a feasible BSP solution exists, the algorithm terminates after finitely many steps. Indeed, since the domain of MP variables $(\mathbf{y}, \mathbf{z}, \mathbf{s})$ is finite, only finitely many subproblems can be defined (and corresponding Benders infeasibility cuts can be generated), so that the optimal value is reached after finitely many steps.

5.1. A branch-and-cut implementation

In this section, we propose an alternative method to the exact algorithm described in the previous section that is generally adopted to solve the Benders reformulation (see Vanderbeck and Wolsey, 2010). The proposed method generates infeasibility cuts into a B&C framework for solving problem MP.

The algorithm has been built within the CPLEX 12.8.0 framework using the CPLEX callback functions. Through these functions, the programmer can almost completely customize the general approach embedded into CPLEX. For example, one can choose the next node to explore in the enumeration tree, choose the branching variable, or define a problem-dependent branching scheme, separate and add his own cutting planes, apply his own heuristic methods, etc. For additional details about the use of the different callback functions, the reader is referred to the documentation of the CPLEX callable library (IBM CPLEX, 2018). The most important implementation issues are discussed below.

5.1.1. Infeasibility cuts

To deal with the master problem MP, function `LazyConstraintCallback` provided by CPLEX is used to separate the infeasibility cuts and add them to MP. Whenever an integer solution is obtained by solving the LP-relaxation of the master problem at a node, the `LazyConstraintCallback` is invoked by CPLEX to examine its feasibility. That is, when the LP solution $(\mathbf{y}, \mathbf{z}, \mathbf{s})$ of the master problem MP is integral, the slave problem SP is solved to verify its feasibility (see Section 5.2). If problem SP is feasible, then a feasible solution of the BSP has been found. Otherwise, the infeasibility cut associated with solution $(\mathbf{y}, \mathbf{z}, \mathbf{s})$ is added to the master problem using function `LazyConstraintCallback`.

Generally, integer solutions $(\mathbf{y}, \mathbf{z}, \mathbf{s})$ can only be found at the leaf nodes of the enumeration tree associated with problem MP. To improve the convergence of the method, we generate infeasibility cuts also at the nodes where the solution is not necessarily integer. To this end, we use function `UserCutCallback` to separate the feasibility cuts and add them to the master problem. Whenever an LP-relaxation is solved and the node is not pruned, the `UserCutCallback` is invoked. If $\mathbf{s}^k$ is integral for some $k \in K$, the `UserCutCallback` separates infeasibility cuts by examining whether a feasible route exists for vehicle k given $\mathbf{s}^k$. In this way, the feasibility cuts can be separated early in the search process and help to strengthen the LP-relaxation of the master problem. Further, additional infeasibility cuts can be added based on the path elimination inequalities (9) and (10). Indeed, given $k \in K$, let $D = \{i \in P : \bar{s}_i^k = 1\}$ be the set of vertices visited by vehicle k. If the subproblem SP

is infeasible, then D is also an infeasible set of vertices, and the infeasible path elimination inequalities (9) and (10) corresponding to D can be added to problem MP.

5.1.2. Valid inequalities

The set of capacity constraints (6) and symmetric breaking constraints (11) are directly added to the initial MP at the root node of the enumeration tree. The lifted cover cuts (8) and the infeasible path elimination cuts (9) and (10) are separated using the `UserCutCallback` function as described in the following.

Whenever an LP-relaxation is solved and the node is not pruned, the values of the variables $\mathbf{z}$ and $\mathbf{s}$ in the LP solution are used to separate the cuts. To separate violated lifted cover inequalities, we identify a cover $C = \{j \in T : z_j^k = 1\}$ such that $\sum_{j \in C} q_j \geq Q_k + 1$ for some $k \in K$ by dynamic programming (DP) (see Nemhauser and Wolsey, 1988). To separate violated infeasible path elimination cuts, we solve subproblem SP as shown in Section 5.2.

5.1.3. Branching and node selection strategy

Based on the results of preliminary experiments, we decided to set a higher branching priority on variables $\mathbf{s}$. As variables $\mathbf{y}$ are safely relaxed to continuous according to Proposition 1, only integer variables $\mathbf{s}$ and $\mathbf{z}$ are considered as integer variables by CPLEX. Because both the infeasibility cuts and the primal heuristic rely on the integer values of variables $\mathbf{s}$, the higher branching priority on variables $\mathbf{s}$ can be helpful in improving both dual and primal bounds, especially in the early exploration of the enumeration tree.

5.1.4. Primal heuristic

With the aim of enhancing the computation of primal solutions during the enumeration tree, we apply a heuristic algorithm using the `HeuristicCallback` CPLEX function.

At a generic node of the B&C tree, we first check if the subproblem SP is feasible, that is, variables $\mathbf{s}$ are integral and feasible routes exist for the subproblem SP. Then the primal heuristic tries to optimally reassign the tourist groups to the vehicles in order to maximize the total profit by solving the following GAP:

$$(GAP) \quad \max \quad \sum_{j \in T} \sum_{k \in K} \bar{p}_j^k z_j^k \tag{14}$$

$$\text{s.t.} \quad \sum_{j \in T} q_j z_j^k \leq Q_k, \qquad \forall k \in K \tag{15}$$

$$\sum_{k \in K} z_j^k = 1, \qquad \forall j \in T \tag{16}$$

$$z_j^k \in \{0, 1\}, \qquad \forall j \in T, \forall k \in K, \tag{17}$$

where $\bar{p}_j^k = \sum_{i \in P} \bar{s}_i^k p_{ji}$ is the profit of assigning group $j \in T$ to vehicle $k \in K$. Problem GAP requires that the capacity of each vehicle cannot be violated (15) and each group must be assigned to exactly one vehicle (16). Problem GAP can be conveniently solved using the MIP solver of CPLEX

using the current solution $\bar{\mathbf{z}}$ of the master problem as a starting solution, and by initializing the CPLEX primal bound as the cost of the best BSP solution found so far.

5.2. Solving the subproblem

Problem *SP* can be decomposed into $|K|$ feasibility subproblems, each one for determining the TW feasibility of a single bus route. The constraints of each subproblem correspond to the well-known traveling salesman problem with TW (TSPTW) (Baker, 1983). For each $k \in K$, solving the subproblem requires to check if a feasible TSPTW solution exists for $\bar{V}_k$, where $\bar{V}_k = \{i \in P : \bar{s}_i^k = 1\}$ denotes the set of vertices visited by vehicle k. This is a known $\mathcal{NP}$-complete problem (Savelsbergh, 1985). Obviously, if $\bar{V}_k = \emptyset$, the corresponding TSPTW is trivially feasible.

Instead of considering the individual feasibility problems, we tackle each of them by considering the objective of minimizing the travel time, as for the classical TSPTW. This approach is best suited for the BSP, since a bus route visiting a set of POIs is obviously preferred over another route visiting the same POIs, but requiring a longer travel time. For every vehicle $k \in K$ such that $\bar{V}_k \neq \emptyset$, we formulate the problem as the following MIP model:

$$\min_{\mathbf{x}^k \in \{0,1\}^{|A|}, \delta^k \in \mathbb{R}_+^{|V|}} \left\{ \sum_{(i,l) \in A} t_{il} x_{il}^k : (\bar{\mathbf{s}}^k, \mathbf{x}^k, \delta^k) \in X_2^k \right\}, \tag{18}$$

where X_2^k corresponds to the set of constraints (2d) involving vehicle k. To solve problem (18), we use the DP-based approach proposed by Baldacci et al. (2012), which has been shown to be particularly effective for the TSPTW. In the following, we briefly describe the method and the reader is referred to Baldacci et al. (2012) for additional details.

For a given vehicle $k \in K$, let $\overline{G} = (\overline{V}, \overline{A})$ be a complete digraph, where $\overline{V} = \bar{V}_k \cup \{p, q\}$, and p and q are two special vertices. A *tour* is defined as a path in $\overline{G}$ starting from vertex p at time e_0, visiting each vertex $i \in \bar{V}_k$ within its TW, and ending at vertex q before l_0. The cost of a tour is equal to the sum of the travel times of the arcs traversed. The TSPTW associated with vehicle k consists of determining the minimum travel time tour, and can be formulated using DP as follows. Define a *forward path* $P = (p, i_1, \ldots, i_k = \sigma(P))$ as an elementary path starting from vertex p at time e_0, visiting vertices $V(P) = \{p, i_1, \ldots, i_k\}$ within their TW, and ending at vertex $\sigma(P)$ at time $t(P)$ with $e_{\sigma(P)} \leq t(P) \leq l_{\sigma(P)}$. Let $\mathscr{P}(S, t, i)$ be the set of all forward paths visiting the subset of vertices $S \subseteq \bar{V}$ and ending at vertex i at time t. In addition, let $g(S, t, i)$ be the minimum travel time path in the set $\mathscr{P}(S, t, i)$. The travel time z(TSPTW) of the optimal TSPTW solution for vehicle k is as follows:

$$z(\text{TSPTW}) = \min_{e_0 \leq t \leq l_0} \{g(\overline{V}, t, q)\}. \tag{19}$$

The $g(S, t, i)$ values can be evaluated using DP as follows: define the state set $\mathscr{S} = \{(S, t, i) : \forall S \subseteq V, \forall i \in S, e_i \leq t \leq l_i\}$. Let $\Omega(t, j, i)$ be the subset of departure times from vertex j to arrive at vertex i at time t, such that $e_i \leq t \leq l_i$, when j is visited immediately before i. $\Omega(t, j, i)$ is defined as follows:

(i) $\Omega(t, j, i) = \{t' : e_j \leq t' \leq \min\{l_j, t - t_{ji}\}\}$ if $t = e_i$, and (ii) $\Omega(t, j, i) = \{t - t_{ji} : e_j \leq t - t_{ji} \leq l_j\}$ if $e_i < t \leq l_i$. The DP recursion to evaluate $g(S, t, i)$ is as follows:

$$g(S, t, i) = \min_{(S', t', j) \in \Psi^{-1}(S,t,i)} \{g(S', t', j) + t_{ji}\}, \quad \forall (S, t, i) \in \mathscr{S}, \tag{20}$$

where $\Psi^{-1}(S, t, i) = \{(S \setminus \{i\}, t', j) : \forall t' \in \Omega(t, j, i),\ \forall j \in \overline{V} \cap (S \setminus \{i\})\}$. The following initialization is required: $g(\{p\}, e_p, p) = 0$ and $g(\{p\}, t, p) = \infty$, $\forall t$, $e_p < t \leq l_p$. The size of the set $\mathscr{S}$ can be huge, but effective dominance and fathoming rules are described in Baldacci et al. (2012) to remove from $\mathscr{S}$ any state (S, t, i) that cannot lead to any feasible or optimal solution, thus speeding up the computation of the optimal TSPTW solution.

6. An iterated local search based metaheuristic

To solve larger BSP instances, we propose an ILS-based metaheuristic algorithm. The algorithm follows the scheme of ILS algorithms (Baxter, 1981; Lourenço et al., 2019) and employs two diversification mechanisms for shaking the incumbent solutions, followed by a `LocalSearch` procedure that improves the modified solution. In particular, the applied diversification mechanisms systematically change the neighborhood, in a manner similar to the VNS (Mladenović and Hansen, 1997; Hansen et al., 2019), which has been effectively applied on vehicle routing problems (see, e.g., Wei et al., 2014, 2015).

Let S denote a feasible BSP solution and $f(S) = M \cdot p(S) - r(S)$ where $p(S)$ is the total profit collected by S, $r(S)$ is the total travel time required by the solution S and M is a very large positive value. The structure of the proposed ILS algorithm is provided in Algorithm 2.

The proposed method starts by constructing an initial solution S, which is improved by the following iterative procedure: Method `Shake` applies one of the two proposed diversification mechanisms (according to argument h), to produce the modified solution S'. Then, S' is improved by the proposed `LocalSearch` method, which employs classical routing and packing LS operators. In addition, it is equipped with MIP-based operators that are used when locally optimal solutions with respect to the classical operators are encountered. Let S'' denote the improved solution produced by the `LocalSearch` method. If $f(S'') > f(S)$, the incumbent solution is set to S'' and the diversification mechanism to be applied on the next iteration is set to 1 ($h \longleftarrow 1$). Otherwise, the method continues by switching the diversification operator for the next iteration. The proposed ILS framework is terminated when a total CPU time T_{max} is completed, or when I_{max} consecutive nonimproving iterations are performed. In the following paragraphs, the various components of Algorithm 2 are described.

6.1. Initial solution

The construction of a feasible BSP solution requires decisions on two levels: (i) assignment of tourist groups to buses and (ii) POI selection and optimal routing for the buses.

Algorithm 2. ILS for the BSP

```
Construct an initial solution S;
h ← 1 ; // neighborhood index
i ← 0 ; // number of non-improving iterations
while (CPU time < T_max) and (i < I_max) do
    S' ← Shake(i, h, S);
    S'' ← LocalSearch(S');
    // neighborhood change
    if f(S'') > f(S) then
        S ← S'' ; // make a move
        h ← 1 ; // change to the first neighborhood
        i ← 0;
    else
        h ← h + 1 ; // change to next neighborhood
        if h > h_max then
            h ← 1 ; // revert to the first neighborhood
        end
        i ← i + 1;
    end
end
return S;
```

We use a simple heuristic algorithm to generate an initial solution with the aim of quickly computing a feasible solution as a starting point for the ILS. The heuristic performs two main steps: (i) assigning of tourist groups to buses by disregarding the profit values and (ii) determining bus routes and the POIs to be visited by also taking into account the profit values. The two steps are as follows:

(i) Due to the difficulty of obtaining a feasible assignment in some instances with limited buses, the proposed construction heuristic starts off by assigning tourist groups to buses, thus when this assignment is decided no routing decisions have been made. As a result, the profit objective is not taken into account and the algorithm constructs feasible tourist assignments to buses with respect to the bus capacity constraints, by applying the best-fit decreasing (BFD) heuristic for this 1D bin packing problem. As a safeguard, if BFD fails to produce a feasible assignment due to tight capacity constraints, the GAP models (14)–(17) are solved to optimality by CPLEX with unit profit values.

(ii) Once the tourist groups have been assigned to buses, the constructive method determines a feasible route for each bus. For bus $k \in K$, the profit of each vertex $i \in P$ is computed as

$\bar{p}_i^k = \sum_{j \in T_k} p_{ji}$, where $T_k \subseteq T$ denotes set of tourist groups assigned to bus k. The bus–vertex pairs $(k,\ i)$, $k \in K$, $i \in T$, are sorted in descending profit $\bar{p}_i^k$. Then, the construction algorithm iteratively selects the bus–vertex pair (k, i) with the highest profit, to insert vertex i into the route traveled by bus k. An insertion is made, if the POI TW and the maximal POI visit constraints are satisfied. If for a POI i multiple feasible positions exist within the route traveled by bus k, vertex i is inserted in the position yielding the minimal travel time increase.

It is worth noting that alternative rules are possible to assign POIs to buses and that, in our experience, the quality of the initial solution is not a key factor for the algorithmic performance. Moreover, regarding the importance of handling the profits, our packing and MIP-based moves described in the following are indeed aimed at improving the groups to POI assignments.

6.2. Diversification operators

The proposed algorithm employs two diversification operators, which as previously mentioned are invoked by method `Shake`. Both operators apply drastic solution modifications, to drive the search away from local optima encountered. The strength of this behavior is controlled by parameter α, which after tuning experiments was set to

$$\alpha = \max\{5, \min\{0.05 \times i, 0.6\} \times (|T| + |P|)\}, \tag{21}$$

where i is the number of consecutive nonimproving iterations currently recorded. The role of parameter α is twofold: (a) it strengthens the diversification effect of both operators when the algorithm consistently fails to identify new improved solutions and (b) it eliminates excessive diversification that would drive the search to extremely poor quality solution regions.

In the following, the two diversification operators are described:

- *RandomMoves* operator randomly performs a sequence of packing or routing moves presented in Section 6.3. To perform each of these moves, one of the seven move types is randomly selected with all move types sharing the same selection probability. Then a move defined by the selected operator is randomly applied, if it respects the model constraints.
- *Destroy&Repair* operator removes tourist groups and vertices randomly from the solution. It then reconstructs this partial solution by first reinserting the removed tourist groups to buses. These insertions are iteratively performed: At each iteration, the profit of assigning every removed group j to every bus $k \in K$ is $\bar{p}_j^k$. If an assignment violates the bus capacity constraints, the associated profit is set to $-\infty$. A regret value $rv_j = \bar{p}_j^{k_1} - \bar{p}_j^{k_2}$ is then evaluated for every unassigned tourist group j, where k_1 and k_2 denote the buses "offering" the highest and second highest profit for group j, respectively. The method identifies the tourist group maximizing the regret value and assigns it to the bus maximizing the aforementioned profit value. Once all tourist groups have been reassigned to buses, POI vertices are inserted into the solution via the mechanism of POI vertex insertion described for the initial solution construction method (Section 6.1).

6.3. Local search

In the BSP, two interconnected decision levels must be jointly tackled: assignment of tourists to buses and routing of buses to the various attractions. With the aim of improving a BSP solution based on both decisions, the LS procedure in the ILS uses various packing moves to optimize the assignment of tourists to the POIs, and routing moves to improve the routes.

The LS procedure first employs two basic types of moves: (i) packing moves that modify the assignments of tourist groups to buses and (ii) routing moves that modify the bus routes. If the basic move types fail to identify an improving solution, two rich MIP-based moves are applied. If the MIP-based moves succeed in improving the incumbent solution, the method continues by reapplying the basic moves in a cyclic framework. On the contrary, if the MIP-based moves fail to generate an improving solution, the `LocalSearch` method terminates by returning the locally optimal solution with respect to the basic operators, as well as the MIP-based moves. In the following, all employed LS operators are described.

6.3.1. Packing moves

By fixing the bus routes of a given BSP solution, BSP reduces to a GAP model that calls for the optimal assignment of tourists to buses. The following moves deal with these assignments. Note that they are only applied if they produce improving and feasible BSP solutions.

- *Group relocation moves* relocate a tourist group from the bus currently assigned to, to another bus.
- *Group swap moves* swap the buses offering service to a tourist group pair. Obviously, the group swap moves involve only tourist groups that are assigned to different buses (in the incumbent BSP solution).

Figure 2 shows an illustrative example of packing moves based on the BSP instance given in Fig. 1.

6.3.2. Routing moves

By fixing the tourist groups to bus assignments of a given BSP solution, BSP reduces to a TOPTW variant, where the profit collection depends on the bus routes. In contrast to the basic TOPTW version, each vertex can be visited multiple times limited by L under the BSP model. The LS operators used for applying routing modifications to a BSP solution are based on the ones given by Hu and Lim (2014) for the TOPTW and are presented in the following. As already stated for the packing moves, they are applied only if they produce feasible and improving BSP solutions.

- *Vertex insertion moves* insert a POI vertex into a bus route. Obviously, the inserted POI must not be already present in the bus route involved.
- *Vertex exchange moves* replace a POI vertex visited by a bus with another POI. First, the removed POI vertex is taken out of the bus route, and then the new POI is inserted into the route position yielding the minimal travel time increase.
- *Vertex relocation moves* relocate a POI vertex visited by a bus to another position in the same bus or to another bus.

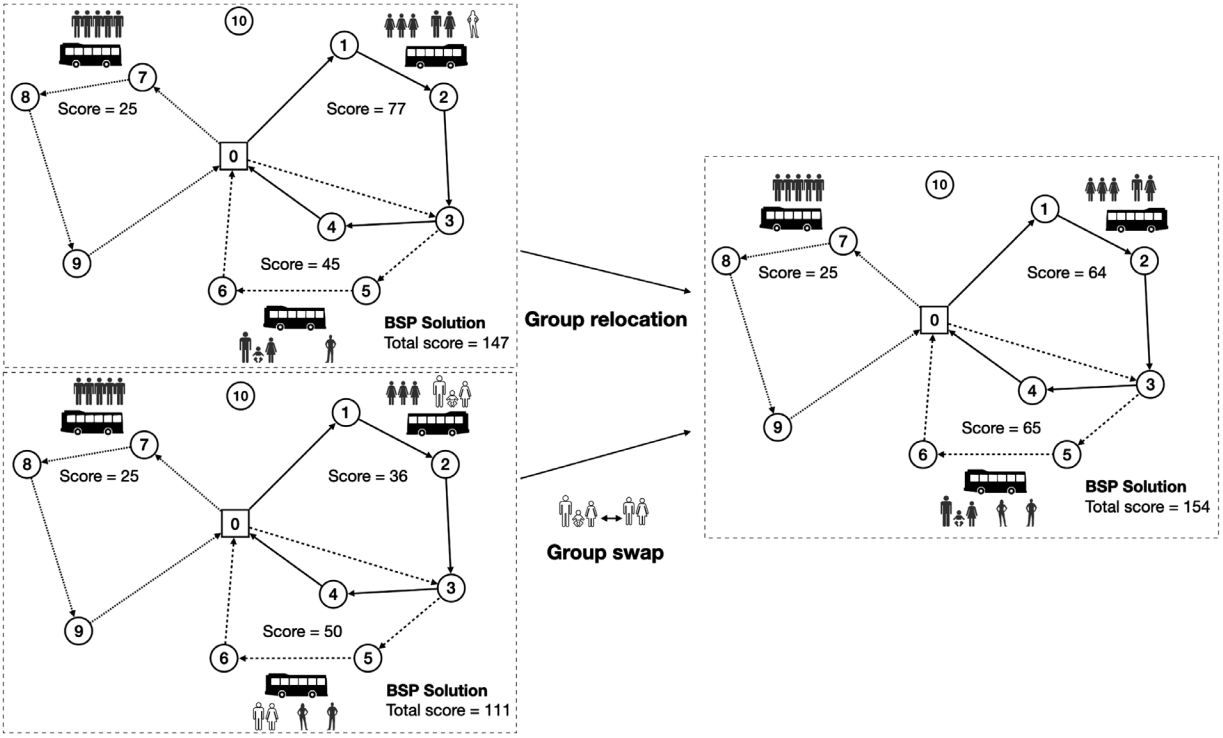

Fig 2. An example of packing moves.

- *Vertex swap moves* swap the visit positions for a pair of POI vertices. This modification can be applied within a single bus or between a bus pair.
- *2-opt* moves replace a pair of arcs from the solution. If both arcs involved belong to the same bus route, this is done by reversing the route segment, lying between the replaced arcs. Otherwise, each of the two bus routes involved in the move are split in their beginning and terminating parts. The beginning part of the first route is connected to the terminating part of the second and vice versa. Note that the last three operators are widely used for routing problems and detailed descriptions are given in Zachariadis and Kiranoudis (2011).

  The moves defined by the seven presented basic move types are evaluated in a random order. The proposed algorithm applies the first improving move which leads to a feasible BSP solution. Note that as explained earlier an improving solution either increases the solution profit or reduces the total travel time if the profit is not modified.

6.3.3. MIP-based moves

If no improving solution can be identified by the above-presented basic LS operators, or in other words when the incumbent solution is locally optimal with respect to the basic operators, two MIP-based moves are performed. Inherently, these moves can directly apply wider solution modifications compared to the basic operators.

As illustrated below, the MIP-based moves are executed with a specific time limit, and they are applied only if no improving solution can be identified by the basic LS operators. Therefore, packing moves are also important to improve the solution. In addition, the MIP-based moves use a starting point for the MIP solver as the incumbent solution and the incumbent lower bound for pruning, hence the packing moves can reduce the computational effort of the MIP solver.

The first MIP-based move solves GAP models (14)–(17), by fixing all bus routes to their current state. For a given vehicle $k \in K$, the profit of serving group $j \in T$ is computed as $\bar{p}_j^k = \sum_{i \in P_k} p_{ji}$, where $P_k \subseteq P$ is the set vertices visited by bus k. The GAP model is solved by CPLEX setting a run time limit of 10 seconds. If a feasible and improved solution is obtained, the LS procedure resumes to the basic moves. Otherwise, the following second MIP-based move is executed.

The second MIP-based move optimizes bus routes, by fixing tourist group assignments to their current state. If the maximal number of POI visits L does not exceed the number of buses in use, the problem can be decomposed into a set of OPTW models, one for each vehicle. For vehicle $k \in K$ currently carrying the tourist group set $T_k \subseteq T$, the profit collected when visiting POI $i \in P$ is $\bar{p}_i^k = \sum_{j \in T_k} p_{ji}$. Then, the OPTW associated with vehicle k is formulated as follows:

$$\text{OPTW}(k) \quad \max \quad \sum_{i \in P} \bar{p}_i^k s_i^k \tag{22}$$

$$\text{s.t} \quad s_i^k \in \{0, 1\}, \qquad \forall i \in P, \tag{23}$$

$$(\mathbf{s}^k, \mathbf{x}^k, \delta^k) \in X_2. \tag{24}$$

The OPTW models are solved by a variant of the B&C implementation of the Benders decomposition method described in Section 5.1, which uses the infeasibility cuts and the `LazyConstraintCallback` function. To separate the infeasibility cuts, the subproblem is solved as described in Section 5.2.

On the contrary, if the number of buses in use exceeds the maximal number of POI visits L, we solve a TOPTW variant formulated as follows:

$$\text{(TOPTW)} \quad \max \quad \sum_{k \in K} \sum_{i \in P} \bar{p}_i^k s_i^k \tag{25}$$

$$\text{s.t} \quad \sum_{k \in K} s_i^k \leq L, \qquad \forall i \in P, \tag{26}$$

$$s_i^k \in \{0, 1\}, \qquad i \in P, \tag{27}$$

$$(\mathbf{s}, \mathbf{x}, \delta) \in X_2. \tag{28}$$

The TOPTW is also solved by a variant of the B&C implementation of the Benders decomposition method (Section 5.1).

The OPTW model for each vehicle, as well as the TOPTW model are both NP-hard problems, which are difficult to be solved to optimality within short computational times. To embed the MIP-based moves that repeatedly solve the aforementioned problems through the search process, CPLEX solver was used with a CPU time bound of 10 seconds. This bound is proved sufficient

in improving the incumbent solution, as described in the computational experiments. In addition, to speed up the TOPTW moves, the profit of the incumbent solution (to be improved by the MIP-based moves) was used as the primal bound to prune unpromising branches in the `BranchCallback` function of CPLEX.

The application of MIP-based moves requires repetitively solving the relevant GAP, OPTW, and TOPTW models. To improve the efficiency of the proposed metaheuristic, it is essential to avoid any duplicate solver calls for the same model instance. To achieve this, the algorithm keeps track of the individual model instances that have been solved and filters out any unnecessary solver calls. This is done by caching the profit vectors that uniquely define a specific instance of the three models, when this instance is solved for the first time.

As already stated, if the MIP-based moves manage to produce a feasible and improving solution, the LS procedure resumes with the application of the basic moves, in a cyclic manner. Otherwise, the LS procedure is terminated.

7. Computational experiments

This section reports computational experiments performed with the proposed Benders decomposition and ILS algorithms on newly generated BSP benchmark instances of diverse characteristics. We present extensive experimental analysis with three main aims. First, we evaluate the performance of our algorithms. Second, we conduct experiments to measure the contribution of each main component of the algorithms. Finally, we evaluate the impact of different profit patterns on the obtained solutions.

Both methods were coded in C++ and executed on a single core of an Intel(R) Core(TM) CPU i7-6700 processor clocked at 3.40 GHz. To solve the various MIP problems, IBM ILOG CPLEX 12.8.0 (IBM CPLEX, 2018) was used. Both the newly introduced BSP benchmark instances and detailed computational results are available at http://www.computational-logistics.org/orlib/bsp.

7.1. Benchmark instances

As mentioned in the introduction, the BSP is a new VRPP variant, and no benchmark instances are available in the literature. Moreover, our problem is also motivated by practical TTDPs, but no real-world BSP data are available. Therefore, we generate a new set of instances based on our experience on related practical applications and on the existing literature (Vansteenwegen et al., 2019b). In particular, we focus on the definition of the profit values to analyze the impact of different profit patterns. Toward this aim, different configurations of the graph, tourist groups, profit values, and bus fleet, were used, as described in the following.

Four distinct sets of POI locations were considered to represent four distinct graphs. The sizes of these sets are 10, 20, 30 and 40. Each graph is generated by randomly generating the depot and POIs coordinates in the Euclidean plane. The travel time matrix $[\hat{t}_{il}]$ is generated by setting $\hat{t}_{il} = \lfloor d_{ij} + 0.5 \rfloor$, where d_{ij} is the Euclidean distance between vertex i and j. For each of these four graphs, two TW configurations were considered: the first one corresponds to four-hour tours and the second one to eight-hour tours. Specifically, for the bus station TW, $e_0 = 0$ and l_0 are set to

Table 1
Configurations of tourist groups and bus fleet

Number of tourists groups	Vehicle type (VT)	Number of buses × Capacity
30	1	4 × 30 people
(90 people)	2	2 × 60 people
	3	4 × 30 people and 1 × 60 people
60	1	6 × 30 people
(180 people)	2	3 × 60 people
	3	3 × 30 people and 2 × 60 people

the maximum duration of a complete bus tour (240 minutes or 480 minutes). The TWs and service times of POIs were randomly generated. Thus, in total eight different network configurations were generated. Given a network, the following information is defined in order to construct a complete BSP benchmark instance: tourist groups, bus fleet, and profit table. In terms of the groups, two tourist group sizes were used: 30 groups and 60 groups. The number of people within each group was generated randomly so that the 30-group problems involve 90 people and the 60-group problems involve 180 people. Values q_j are integers in the interval [1,5]. Regarding the bus fleet, we have used three different configurations for the tourist group sizes considered. These configurations are reported in Table 1. Thus, in total 48 test problems without considering the profit table were generated.

A profit table $[p_{ji}]$ defines the profit enjoyed by tourist group $j \in T$ visiting POI $i \in P$. Three different profit tables were generated for each graph and tourist group pair. The rationale was to capture cases of random, dissimilar, and similar preferences of tourists among the POIs of a touristic destination. These three profit tables are called PT, PT-D, PT-S, respectively.

- *PT* (*uniform profits*): In this matrix, about 30% of the profit values are set to 0 and the rest of the profit values have been uniformly set into a predefined interval. Then, the various profit values are normalized, so that $\sum_{i \in P} p_{ji} = 100, \forall j \in T$. This is to ensure that all tourist groups are equally considered in terms of their POIs preference.
- *PT-D* (*dissimilar profits*): For each $j \in T, i \in P$ pair, the profit enjoyed by tourist group j when visiting POI i is set to $p_{ji} = sin(\theta(j) + \phi(i))$, where $\phi(i) = 2i\pi/|P|$ and $\theta(j) = 2j\pi/|T|$. If $p_{ji} < 0$, then we set $p_{ji} = 0$. Again, the final profit matrix is obtained by normalizing the various $p_{ji} > 0$ values, so that the total profit enjoyed by each group over all POIs is equal to 100. Using this strategy, contrasting POI preferences are promoted among tourist groups.
- *PT-S* (*similar profits*): For the first tourist group $j = 0$, we randomly generate uniformly distributed profit values. Given the first row of the profit matrix, the profit values for the rest of the rows $p_{ji}, 0 < j < |T|, 0 \leq i < |P|$ are uniformly distributed within a symmetric interval around p_{0i}. Finally, the resulting profits are normalized, as earlier described. This class of profit matrices reflects tourists with similar POI preferences.

For each of the 48 test cases, 3 profit matrices are generated as per the abovementioned classes, resulting in a total of 144 instances. Finally, the maximum allowed visits per POI L takes values from $\{1, 3, 5, +\infty\}$, where $L = +\infty$ means that no restrictions are imposed on the number of visits

of each POI. Thus, in total 576 (i.e., 144 × 4) complete BSP test instances were generated and solved by the proposed methodologies.

7.2. Results of the proposed Benders decomposition method

Our exact method based on Algorithm 1, called BDM, has been implemented using the B&C solution framework provided by ILOG CPLEX. BDM adds the different valid inequalities, capacity constraints (6), lifted cover cuts (8), infeasible path elimination constraints (9) and (10), by means of the `UserCutCallback` function of CPLEX. Further, BDM also invokes a primal heuristic using the `HeuristicCallback` function. To analyze the effectiveness of the different components of BDM, we compare the following variants by computational tests.

- BDM-1: only capacity constraints (6) and function `HeuristicCallback` (HEU) are used, that is, `UserCutCallback` is not used and thus neither lifted cover cuts nor infeasible path elimination constraints are separated, but `HeuristicCallback` is enabled.
- BDM-2: only capacity constraints (6), lifted cover cuts (8), and HEU are used, that is, only lifted cover inequalities are separated in the `UserCutCallback`, and also `HeuristicCallback` is used. No infeasible path elimination constraints are separated.
- BDM-3: only capacity constraints (6), lifted cover cuts (8), and infeasible path elimination constraints (9) are used, that is, both lifted cover cuts and infeasible path elimination constraints are separated in the `UserCutCallback`, but `HeuristicCallback` is not used.
- BDM: complete version of the exact method, where capacity constraints (6), lifted cover cuts (8), infeasible path elimination constraints (9) and (10), and HEU are used, that is, both lifted cover cuts and infeasible path elimination constraints are separated in the `UserCutCallback` and `HeuristicCallback` is used.

The four implementations are tested on instances with 30 tourist groups and up to 20 POIs. A time limit of 7200 seconds was imposed on each run. Further, in our experiments, all CPLEX cuts are switched on.

Table 2 shows the results of versions BDM-1, BDM-2, and BDM, whereas Table 3 reports the results of BDM-3 and BDM. In these tables, each row gives the results on eight instances corresponding to two networks graphs and four values of parameter L. Each test set is identified by three columns $|P|$, VT, and PT, which represent the number of POIs, the type of bus fleet (see Table 1), and profit matrix, respectively. In both tables, the percentage gap between the obtained primal LB and dual bound UB is evaluated as $Gap = 100.0 \times \frac{UB-LB}{UB}$. Columns $\#opt$, $g(\%)$, $\#node$, t (seconds) under each algorithm version indicate the number of instances solved to optimality, the average percentage gap, the average number of nodes explored by the BC algorithm, and the average computing time in CPU seconds computed over the instances solved to optimality, respectively.

The results reported in Table 2 show the effectiveness of the valid inequalities. Indeed, using the lifted cover inequalities, BDM-2 is able to obtain two more optimal solutions compared to the BDM-1 version, whereas the average gap of the dual bound is also improved. For the instances solved to optimality by both BDM-1 and BDM-2 versions, BDM-2 significantly reduces the

Table 2
Performance analysis on lifted cover cuts and infeasible path elimination constraints of BDM with $|T| = 30$

			BDM-1				BDM-2				BDM			
$\|P\|$	VT	PT	*#opt*	*g*(%)	*#node*	*t* (seconds)	*#opt*	*g*(%)	*#node*	*t* (seconds)	*#opt*	*g*(%)	*#node*	*t* (seconds)
10	1	PT-D	8	0.00	7220.4	21.5	8	0.00	5530.1	19.3	8	0.00	4129.3	24.0
		PT-S	2	5.56	413,866.6	5424.9	2	5.51	388,157.1	5412.4	5	1.85	96,351.5	2748.3
		PT	2	5.81	437,362.6	5414.6	2	5.13	309,814.6	5414.0	5	1.28	178,248.3	2938.0
	2	PT-D	8	0.00	342.4	0.7	8	0.00	269.0	0.6	8	0.00	217.6	1.0
		PT-S	5	3.67	345,679.4	2745.8	5	3.66	405,371.1	2706.1	8	0.00	576.5	4.7
		PT	8	0.00	26,460.1	144.3	8	0.00	2796.1	6.4	8	0.00	638.4	2.9
	3	PT-D	8	0.00	1725.4	4.4	8	0.00	1194.4	4.0	8	0.00	890.5	4.7
		PT-S	5	4.69	287,819.0	2715.4	5	4.75	296,680.4	2721.9	8	0.00	8595.9	104.3
		PT	2	5.67	413,819.4	5401.3	3	4.54	346,244.1	5062.5	8	0.00	12,382.1	71.2
20	1	PT-D	0	36.28	222,598.3	7200.3	0	37.11	214,690.6	7200.4	0	30.97	155,677.3	7200.2
		PT-S	0	41.04	266,417.6	7200.2	0	40.97	233,475.9	7200.2	0	32.67	87,307.3	7200.2
		PT	0	49.20	274,042.5	7200.2	0	48.61	214,220.3	7200.3	0	45.81	136,807.0	7200.2
	2	PT-D	4	22.90	299,132.8	3605.5	4	22.72	250,837.3	3605.7	4	17.58	69,901.5	3610.2
		PT-S	0	33.43	357,732.8	7200.1	0	31.97	304,740.4	7200.1	4	14.14	54,157.6	3626.7
		PT	4	28.60	229,439.0	3696.0	4	27.10	180,092.9	3635.4	4	21.47	103,456.0	3628.4
	3	PT-D	4	25.70	234,984.6	4587.3	4	23.11	197,284.0	4180.8	4	15.95	122,127.3	4709.5
		PT-S	0	38.04	328,218.9	7200.2	0	33.90	267,429.8	7200.2	2	15.21	113,111.9	6151.3
		PT	0	40.04	361,030.0	7200.3	1	38.21	318,780.9	7021.7	1	30.52	163,867.0	6675.5
All			60	18.92	250,438.4	2813.5	62	18.18	218,756.0	3060.8	85	12.64	72,691.3	2286.7

number of nodes explored. The results obtained by the complete BDM version shows that the two sets of infeasible path elimination cuts (9) and (10) are also effective. Indeed, BDM is able to solve 23 more test instances to optimality compared to BDM-2. Moreover, the average percentage gap is reduced by 5.54%, and the average number of nodes is also reduced by 66.8%. It is worth noting that all CPLEX cuts were used in all the different BDM versions and, in particular, in version BDM-1. Hence, the addition of the new cuts (8), (9), and (10) in the other BDM versions also achieves improvements with respect to the CPLEX cuts.

The results reported in Table 3 indicate that the use of `HeuristicCallback` is effective in improving the primal bounds, and the overall performance of BDM. With the use of the heuristic callback, BDM outperforms BDM-3, both on the number of instances solved to optimality and on the quality of the final bounds. The main advantage of using the heuristic callback is that BDM is capable of identifying good primal bounds at the early stages of the enumeration tree exploration.

Tables 2 and 3 show that the effectiveness of the exact method owes to the proposed valid inequalities and the primal heuristic, which are helpful in improving the upper and lower bounds, respectively.

Table 4 reports detailed results of the BDM version on instances with up to 20 POIs. The following columns are reported in the table: the number of instances of the corresponding group of instances (*#inst*), *number of instances solved to optimality* (*#opt*), the average percentage gap at the root node of the enumeration tree (g_{root} (%)), the final average percentage gap (g (%)), the average computing time (t (seconds)), the average time spent for the lazy constraint callback (t_{lazy}), the

Table 3
Performance analysis on heuristic callback of BDM with $|T| = 30$

			BDM-3				BDM			
$\|P\|$	VT	PT	*#opt*	*g*(%)	*#node*	*t* (seconds)	*#opt*	*g*(%)	*#node*	*t* (seconds)
10	1	PT-D	8	0.00	12902.9	50.3	8	0.00	4129.3	24.0
		PT-S	5	2.45	126,216.0	2757.6	5	1.85	96,351.5	2748.3
		PT	5	2.46	198,124.6	2979.0	5	1.28	178,248.3	2938.0
	2	PT-D	8	0.00	427.0	0.9	8	0.00	217.6	1.0
		PT-S	8	0.00	712.9	4.2	8	0.00	576.5	4.7
		PT	8	0.00	1753.4	4.1	8	0.00	638.4	2.9
	3	PT-D	8	0.00	1982.1	6.9	8	0.00	890.5	4.7
		PT-S	8	0.00	7814.9	79.0	8	0.00	8595.9	104.3
		PT	5	1.69	358,385.3	2713.4	8	0.00	12,382.1	71.2
20	1	PT-D	0	33.52	171,660.4	7200.2	0	30.97	155,677.3	7200.2
		PT-S	0	33.03	96,923.0	7200.2	0	32.67	87,307.3	7200.2
		PT	0	47.45	149,452.1	7200.2	0	45.81	136,807.0	7200.2
	2	PT-D	4	19.31	82,959.1	3606.4	4	17.58	69,901.5	3610.2
		PT-S	4	15.58	52,458.6	3625.9	4	14.14	54,157.6	3626.7
		PT	4	23.08	95,087.4	3628.0	4	21.47	103,456.0	3628.4
	3	PT-D	1	25.32	196,756.0	6361.0	4	15.95	122,127.3	4709.5
		PT-S	2	22.27	116,714.0	5917.3	2	15.21	113,111.9	6151.3
		PT	1	32.51	197,168.0	6755.4	1	30.52	163,867.0	6675.5
All			79	14.37	103,749.9	2566.0	85	12.64	72,691.3	2286.7

average time spent for separating lifted cover cuts (t_{cover}), the average time spent for separating infeasible path elimination constraints (t_p), the average time for the heuristic callback function (t_h), the average time spent for solving the TSPTW subproblems (t_{TSPTW}), and the average number of lifted cover cuts and infeasible path elimination constraints identified (*#cover* and *#IPEC*, respectively). All computing times are given in seconds.

The results of Table 4 indicate that the BSP is a challenging VRP with profits, and that the generated set of benchmark instances represent difficult BSP test cases. Indeed, the average percentage gap at the root node is about 70%. However, the final average gap obtained by BDM is significantly reduced to about 16%, demonstrating the effectiveness of the proposed method. Regarding the different valid inequalities, the table shows that several lifted cover cuts and infeasible path elimination constraints are identified. Their use is effective not only at the root node but also during the enumeration tree. However, the corresponding separation times are relevant as shown by columns t_{cover} and t_p. BDM computes optimal solutions for 135 of 288 instances. The CPU time results show that both the heuristic and the TSPTW subproblems can be efficiently solved.

Table 5 gives an overview of the results of Table 4 as a function of the profit tables and the vehicle types. The table reports the total number of instances solved to optimality (*#opt*) and the average of the final gaps computed (*g*(%)) for each main parameter. The overview shows that the profit tables based on the uniform profits (variant PT) are more challenging for our method, as testified by the number of instances solved to optimality (37 over 96 instances) and the final average percentage gap equal to 21.5%. Regarding the different vehicle types, the table shows that instances with lower vehicle capacities (capacity equal to 30 people) are more difficult than

Table 4
Detailed result of BDM on small-sized instances

P	$\|T\|$	VT	PT	#inst	#opt	g_{root} (%)	g (%)	#node	t (seconds)	t_{lazy}	t_{cover}	t_p	t_{tsptw}	t_h	#cover	#IPEC
10	30	1	PT-D	8	8	64.33	0.00	4129.3	24.0	0.1	5.3	2.8	0.3	0.2	86.3	675.1
			PT-S	8	5	62.58	1.85	96,351.5	2748.3	4.5	258.4	306.5	8.6	3.4	5840.5	69,025.1
			PT	8	5	55.34	1.28	178,248.3	2938.0	2.3	348.8	345.7	10.8	7.2	5137.5	85,107.1
		2	PT-D	8	8	43.40	0.00	217.6	1.0	0.1	0.3	0.2	0.1	0.0	3.6	52.0
			PT-S	8	8	55.56	0.00	576.5	4.7	0.1	1.1	0.6	0.2	0.0	30.6	280.0
			PT	8	8	48.25	0.00	638.4	2.9	0.1	1.0	0.5	0.2	0.0	22.5	223.0
		3	PT-D	8	8	54.54	0.00	890.5	4.7	0.1	1.2	0.6	0.2	0.1	30.8	167.9
			PT-S	8	8	39.02	0.00	8595.9	104.3	0.8	16.0	14.9	0.8	0.3	944.3	5074.3
			PT	8	8	52.29	0.00	12,382.1	71.2	0.5	19.1	13.4	0.8	0.5	315.3	5179.0
10	60	1	PT-D	8	4	70.02	4.26	81,592.1	4325.6	0.5	535.1	487.5	3.4	9.5	5204.6	19,841.5
			PT-S	8	2	56.60	5.65	50,479.0	5534.1	2.2	413.3	446.3	3.5	5.3	4957.8	15,555.1
			PT	8	1	69.06	12.72	90,724.9	6635.6	1.2	663.0	589.0	4.0	9.4	4291.9	22,546.6
		2	PT-D	8	8	64.23	0.00	734.6	13.2	0.2	2.9	1.4	0.2	0.1	53.5	178.4
			PT-S	8	8	59.99	0.00	830.6	29.5	0.1	3.0	1.4	0.1	0.2	53.6	154.3
			PT	8	8	60.87	0.00	13,268.8	234.6	0.4	53.9	41.5	0.8	0.7	466.1	5518.8
		3	PT-D	8	8	67.12	0.00	53,164.4	1462.9	0.5	202.2	73.1	1.0	5.3	1125.0	4215.3
			PT-S	8	4	61.38	1.02	66,585.8	4437.1	1.9	350.4	354.2	3.8	8.4	3133.0	17,465.3
			PT	8	2	66.76	7.88	99,069.3	5478.4	2.0	673.7	645.4	5.4	12.7	1743.5	35,203.1
20	30	1	PT-D	8	0	83.56	30.97	155,677.3	7200.2	13.4	563.7	1079.1	134.8	11.9	3663.5	90,567.9
			PT-S	8	0	87.98	32.67	87,307.3	7200.2	12.6	365.9	791.0	116.0	7.5	2383.5	66,002.3
			PT	8	0	92.48	45.81	136,807.0	7200.2	9.1	579.9	1103.9	122.5	11.3	2237.4	10,9731.6
		2	PT-D	8	4	79.17	17.58	69,901.5	3610.2	266.4	639.0	1308.4	430.9	3.1	4243.8	30,1860.3
			PT-S	8	4	79.93	14.14	54,157.6	3626.7	141.4	515.6	1025.5	326.1	5.5	8986.4	21,7381.5
			PT	8	4	83.20	21.47	103,456.0	3628.4	57.6	689.1	971.0	252.2	3.4	11,968.0	26,9260.4
		3	PT-D	8	4	80.07	15.95	122,127.3	4709.5	23.1	442.9	832.6	147.3	6.0	3833.4	121,056.3
			PT-S	8	2	80.25	15.21	113,111.9	6151.3	18.5	342.8	621.1	107.4	6.8	3713.4	86,029.0
			PT	8	1	85.77	30.52	163,867.0	6675.5	16.3	513.5	766.7	119.0	12.3	4480.1	113,779.4
20	60	1	PT-D	8	0	81.54	39.11	40,514.3	7200.4	3.8	545.6	720.6	26.4	5.7	1100.9	11,518.1
			PT-S	8	0	87.54	53.07	22,171.0	7200.4	19.6	358.2	585.9	17.0	5.6	1583.1	8825.9
			PT	8	0	88.94	50.65	31,441.5	7200.4	8.1	500.8	927.8	31.4	5.0	980.4	15,196.5
		2	PT-D	8	4	82.10	19.77	84,886.9	6073.3	16.4	746.9	1348.8	117.6	3.9	3842.0	74,393.5
			PT-S	8	1	84.50	28.13	65,872.3	6659.3	12.4	528.3	870.9	59.9	4.8	1777.0	43,563.0
			PT	8	0	88.31	39.19	96,351.4	7200.3	11.6	878.0	1278.7	77.5	11.4	1535.4	73,268.9
		3	PT-D	8	0	84.17	34.27	54,532.0	7200.3	4.8	597.7	613.5	28.3	14.0	1193.7	14,331.5
			PT-S	8	0	81.36	46.55	34,710.0	7200.3	18.3	418.3	519.4	19.7	11.6	1021.8	10,680.8
			PT	8	0	86.49	48.72	47,700.3	7200.3	14.7	545.3	616.9	21.9	11.8	721.3	14,776.3
All				288	135	71.05	16.55	62,740.8	2933.2	19.2	366.2	534.6	62.0	5.5	2614.5	54,585.0

Table 5
Results overview on the small-sized instances

Profit tables	#opt	g(%)	Vehicle types	#opt	g(%)
PT	37/96	21.5	1	25/96	23.2
PT-D	56/96	13.5	2	65/96	11.7
PT-S	42/96	16.5	3	45/96	16.7

instances involving larger vehicles (with capacity up to 60 people), as also testified by the final average percentage gaps.

7.3. Results of the ILS algorithm

This section reports on the computational experiments performed with the use of the proposed ILS metaheuristic. The packing and routing moves (see Section 6.3) are standard solution modification mechanisms that have proven to be effective for BSP-related problems. Using the aforementioned moves as the basis for comparisons, we perform computational experiments to assess the effectiveness of the MIP-based moves. We consider the following versions of the ILS algorithm.

- ILS-1: the basic implementation of the ILS using only packing and routing moves (without MIP-based moves);
- ILS-2: version ILS-1 equipped with GAP moves;
- ILS-3: version ILS-1 equipped with OPTW/TOPTW moves;
- ILS: the complete version of the ILS, equipped with all move types.

The four ILS versions were tested on instances with 20 POIs, 60 tourist groups and a maximum bus tour duration of four hours. Each run was terminated after the completion of 600 CPU seconds, or 200 nonimproving iterations, that is, parameters T_{max} and I_{max} are set equal to 600 and 200 in Algorithm 2, respectively (the values of the parameters were identified as results of preliminary experiments, which were conducted to identify good parameter settings). The obtained results are provided in Table 6. Each row corresponds to a single benchmark instance. For sake of the exposition, we name a benchmark instance as xP-yT-zTW-tVT-lMV-*mat* represents a BSP test case that involves x POIs, y tourist groups, $z \in \{4, 8\}$ maximum tour duration, $t \in \{1, 2, 3\}$ the vehicle type, $l \in \{1, 3, 5, +\infty\}$ maximum visits per POI (L), and *mat* type of profit matrix. Column *Profit* reports the best solution value and column t (seconds) reports the computing time in seconds. For ILS-2, ILS-3, and ILS, column *impr (%)* reports the solution improvement percentage with respect to ILS-1 solution.

For the complete version ILS, column t_{optw}, t_{toptw}, and t_{bpp} provides the computational time required for solving the OPTW, TOPTW, and GAP models (MIP-based moves), respectively.

In terms of the solution profit, it is evident that the incorporation of the MIP-moves leads to significant solution improvements. Incorporating the GAP moves (ILS-2) in the basic scheme (ILS-1) improves the solution quality obtained for 22 of the 36 test cases, with an average improvement of 0.24%. The required CPU times are significantly increased, however the heuristic remains very fast (about four CPU seconds, on average). The solution improvement effect is stronger for the OPTW/TOPTW moves (ILS-3). Compared to the basic scheme (ILS-1), the OPTW/TOPTW moves manage to improve 32 of the 36 solutions, whereas the average improvement is about 3.2%. However, the CPU times required for solving the OPTW and TOPTW models increase the average total run time to about 226 seconds. The complete algorithm version (ILS) outperforms all three other versions, regarding the profit objective for all 36 benchmark instances, with an average total run time of about 202 seconds. This is lower than the ILS-3 version, since

Table 6
Analysis on the main optimization-based moves

	ILS-1		ILS-2			ILS-3			ILS					
Instance	*Profit*	*t* (seconds)	*Profit*	*impr* (%)	*t* (seconds)	*Profit*	*impr* (%)	*t* (seconds)	*Profit*	*impr* (%)	*t* (seconds)	t_{optw}	t_{toptw}	t_{gap}
20P-60T-4TW-1VT-1MV-PT	1374	0.1	1403	2.11	6.4	1379	0.36	17.6	1409	2.55	31.8	0	27.5	4.2
20P-60T-4TW-1VT-1MV-PT-D	1951	0.1	2028	3.95	4.1	2052	5.18	18.4	2052	5.18	12.4	0	9.0	3.3
20P-60T-4TW-1VT-1MV-PT-S	1086	0.1	1120	3.13	16.9	1099	1.20	81.8	1132	4.24	70.3	0	65.9	4.4
20P-60T-4TW-2VT-1MV-PT	1389	0.1	1420	2.23	2.2	1469	5.76	307.1	1485	6.91	414.3	0	412.6	1.6
20P-60T-4TW-2VT-1MV-PT-D	2061	0.1	2093	1.55	1.1	2176	5.58	31.4	2176	5.58	23.6	0	22.6	0.9
20P-60T-4TW-2VT-1MV-PT-S	1269	0.2	1271	0.16	2.5	1269	0	601.8	1271	0.16	601.1	0	600.4	0.7
20P-60T-4TW-3VT-1MV-PT	1493	0.1	1468	−1.67	4.9	1493	0	47.4	1516	1.54	53.4	0	50.7	2.6
20P-60T-4TW-3VT-1MV-PT-D	2175	0.1	2211	1.66	4.4	2211	1.66	19.0	2211	1.66	13.2	0	11.1	2.1
20P-60T-4TW-3VT-1MV-PT-S	1270	0.1	1273	0.24	3.4	1272	0.16	162.6	1276	0.47	117.8	0	115.2	2.4
20P-60T-4TW-1VT-3MV-PT	1633	0.1	1678	2.76	5.5	1631	−0.12	602.5	1638	0.31	474.4	0	471.2	3.1
20P-60T-4TW-1VT-3MV-PT-D	2303	0.2	2344	1.78	12.0	2354	2.21	335.2	2354	2.21	163.4	0	159.8	3.4
20P-60T-4TW-1VT-3MV-PT-S	1394	0.2	1345	−3.52	7.7	1403	0.65	579.6	1379	−1.08	603.9	0	603.7	0.2
20P-60T-4TW-2VT-3MV-PT	1511	0.1	1590	5.23	3.9	1639	8.47	249.1	1631	7.94	173.5	172.4	0	1.0
20P-60T-4TW-2VT-3MV-PT-D	2128	0.2	2071	−2.68	1.7	2176	2.26	6.3	2176	2.26	7.4	6.4	0	0.9
20P-60T-4TW-2VT-3MV-PT-S	1295	0.1	1361	5.10	2.5	1424	9.96	159.0	1424	9.96	142.4	141.5	0	0.8
20P-60T-4TW-3VT-3MV-PT	1718	0.1	1689	−1.69	6.1	1732	0.81	611.8	1667	−2.97	605.7	0	605.6	0.1
20P-60T-4TW-3VT-3MV-PT-D	2253	0.1	2190	−2.80	2.0	2305	2.31	327.7	2312	2.62	236.8	0	234.8	2.0
20P-60T-4TW-3VT-3MV-PT-S	1431	0.1	1434	0.21	2.7	1421	−0.70	602.5	1431	0	608.8	0	608.6	0.2
20P-60T-4TW-1VT-5MV-PT	1645	0.2	1683	2.31	4.7	1734	5.41	603.8	1756	6.75	438.1	0	434.3	3.7
20P-60T-4TW-1VT-5MV-PT-D	2315	0.2	2238	−3.33	3.6	2354	1.68	408.1	2354	1.68	417.2	0	414.2	2.9
20P-60T-4TW-1VT-5MV-PT-S	1371	0.1	1380	0.66	3.4	1416	3.28	608.0	1426	4.01	543.9	0	540.6	3.3
20P-60T-4TW-2VT-5MV-PT	1511	0.1	1590	5.23	3.9	1639	8.47	248.6	1631	7.94	173.2	172.1	0	1.0
20P-60T-4TW-2VT-5MV-PT-D	2128	0.2	2071	−2.68	1.7	2176	2.26	6.3	2176	2.26	7.4	6.4	0	0.9
20P-60T-4TW-2VT-5MV-PT-S	1295	0.1	1361	5.10	2.5	1424	9.96	159.7	1424	9.96	142.6	141.8	0	0.8
20P-60T-4TW-3VT-5MV-PT	1691	0.1	1698	0.41	2.3	1757	3.90	192.3	1749	3.43	129.3	127.2	0	2.0
20P-60T-4TW-3VT-5MV-PT-D	2264	0.1	2211	−2.34	2.3	2316	2.30	5.0	2314	2.21	12.1	9.7	0	2.3
20P-60T-4TW-3VT-5MV-PT-S	1430	0.1	1422	−0.56	2.7	1441	0.77	105.3	1441	0.77	154.0	150.9	0	3.0
20P-60T-4TW-1VT-∞MV-PT	1674	0.2	1713	2.33	8.0	1777	6.15	190.2	1772	5.85	157.0	153.6	0	3.3
20P-60T-4TW-1VT-∞MV-PT-D	2337	0.1	2193	−6.16	3.4	2354	0.73	8.6	2354	0.73	6.2	3.0	0	3.1
20P-60T-4TW-1VT-∞MV-PT-S	1387	0.1	1344	−3.10	4.5	1435	3.46	119.4	1438	3.68	122.0	118.5	0	3.4
20P-60T-4TW-2VT-∞MV-PT	1511	0.1	1590	5.23	3.9	1639	8.47	248.4	1631	7.94	173.1	172.0	0	1.0
20P-60T-4TW-2VT-∞MV-PT-D	2128	0.2	2071	−2.68	1.7	2176	2.26	6.3	2176	2.26	7.4	6.3	0	1.0
20P-60T-4TW-2VT-∞MV-PT-S	1295	0.1	1361	5.10	2.5	1424	9.96	160.1	1424	9.96	142.1	141.3	0	0.8
20P-60T-4TW-3VT-∞MV-PT	1691	0.1	1698	0.41	2.3	1757	3.90	192.4	1749	3.43	129.1	127.0	0	2.1
20P-60T-4TW-3VT-∞MV-PT-D	2264	0.1	2211	−2.34	2.3	2316	2.30	4.9	2314	2.21	12.1	9.7	0	2.3
20P-60T-4TW-3VT-∞MV-PT-S	1430	0.1	1422	−0.56	2.7	1441	0.77	105.7	1441	0.77	153.9	150.8	0	3.0
All	61,101	0.1	61,246	0.24	4.1	63,081	3.24	225.9	63,110	3.29	202.1	50.3	149.7	2.0

Table 7
Analysis on the optimization-based moves: relocation, 2-opt, and packing moves

		ILS-1-a		ILS-1-b		ILS-1-c		ILS-4	
TW	PT	$g(\%)$	$t(\%)$	$g(\%)$	$t(\%)$	$g(\%)$	$t(\%)$	$g(\%)$	$t(\%)$
4TW	PT	−0.28	16.6	0.48	3.2	18.77	45.0	−0.11	20.9
	PT-D	1.12	14.2	0.36	−0.3	35.14	33.5	0.11	37.8
	PT-S	0.69	24.5	−0.50	0.6	6.65	38.3	−0.07	23.2
8TW	PT	1.74	12.8	2.33	5.3	14.06	28.0	−0.08	5.1
	PT-D	1.51	−8.1	0.68	−12.9	35.71	33.1	0.07	27.6
	PT-S	1.99	4.5	1.02	0.7	5.36	20.3	−0.04	2.7
		1.28	10.7	0.86	−0.5	21.63	32.8	−0.01	12.1

in the complete version half of the MIP-based moves correspond to the solution of the easier GAP model.

To further verify the effectiveness of the ILS moves, we also considered the following versions of the basic implementation ILS-1 (see also Section 6.3.1):

- ILS-1-a: without group relocation moves;
- ILS-1-b: without the option of reversing a route segment in the 2-opt moves;
- ILS-1-c: without the packing moves (group relocation and group swap).

To verify the impact of the packing moves on algorithm ILS, we also considered an additional version of the algorithm, denoted as ILS-4, which is obtained by disabling the packing moves.

For these experiments, we considered the instances with $|P| = 20$ and $|T| = 60$. In particular, we analyze the results obtained by grouping the instances based on the type of TW constraints, that is, four-hour tours (named 4TW) and eight-hour tours (8TW), for a total of $36 \times 2 = 72$ instances.

The results obtained are summarized in Table 7. For each version of implementations and group of instances (type of TW and profit matrices "PT"), the table shows the average percentage gaps of the different versions with respect to the basic version ILS-1, that is, $g(\%) = 100 \times (z(\text{ILS-1}) - z(\text{ILS-1-x}))/z(\text{ILS-1})$, $x \in \{a, b, c\}$, where z(ILS-1) and z(ILS-1-x) are the total sum of the profits of the instances in the group for ILS-1 and ILS-1-x, respectively. For version ILS-4, column $g(\%)$ gives the average gaps of algorithms ILS and ILS-4. For all the algorithm versions considered, columns $t(\%)$ report the average percentage gap of computing times. The last line of the table reports the average gaps computed over the different group of instances. In the table, because of the maximization objective, a gap $g(\%)$ greater than zero indicates that the corresponding version shows worse average performance with respect to the reference version. Conversely, regarding columns $t(\%)$, a gap greater than zero indicates that the corresponding version has improved average performance with respect to the reference version.

The results obtained can be summarized as follows:

- The results of version ILS-1-c shows that the packing moves greatly play a key role in improving the quality of the solutions of the basic version ILS-1 and that the additional computational

times is worth it. The moves are particularly useful for profit matrices PT-D showing dissimilar preferences of tourists among the POIs, a case close to the practice.

- The comparison between ILS-1-a and ILS-1-c shows that the main source of the improvements of ILS-1-c can be attributed to the group swap moves. Regarding the group relocation moves, because the size of a group ranges in [1, 5] which is relatively small compared to the vehicle capacity (30 and 60), this type of move could be useful being the group sizes small. Indeed, the group relocation moves are useful if some buses have some empty seats in a BSP solution, that is, if $\sum_{k \in K} Q_k > \sum_{j \in T} q_j$. For example, as shown in Table 1, for the instances with 30 groups, the total number of available seats is a bit larger than the number of people.
- The analysis of version ILS-1-b shows that removing the reversing of a route segment produces on average worse solutions as shown by the average gap equal to about 1%. This is particularly true for the instances with eight-hour tours (8TW) showing weaker TW constraints than the 4TW group of instances.
- The results of version ILS-4 indicate that ILS-4 produces on average slightly improved solutions than ILS, at a slightly lower average computing time. However, it is worth noting that on the profit matrices of type PT-D, ILS-4 obtains on average worse results than ILS, thus showing the importance of using the packing moves for profit matrices of type PT-D.

Table 8 reports a comparison between the exact method BDM and the ILS algorithm on the set of instances involving up to 20 POIs and 30 groups. For sake of the comparison, five ILS executions were made, each using a different seed value for the various random decisions. Each row of Table 8 corresponds to a set of eight benchmark instances (four L values and two bus tour duration limits). Average values over the eight instances are given as follows: Column *Profit* reports the average total profits and column t (seconds) provides the average CPU time in seconds. Since ILS was run five times for each test set, columns *Best profit*, *Avg profit*, *Best impr (%)*, and *Avg impr (%)* provide the total profit of the best solutions found (over the five runs), the average profit value (over the five runs), the percentage of profit improvement of the best ILS solutions with respect to the BDM solutions, the percentage of profit improvement of the average ILS solutions (over the five runs) with respect to the BDM solutions.

The obtained results demonstrate that the proposed ILS method is effective for the BSP model. For the instances with just 10 POIs (the majority of which were optimally solved by BDM) ILS matched or improved the obtained BDM solutions. ILS improves BDM solution objectives for 8 of 18 test sets and the average improvement is about 0.09%. In addition, ILS appears to be significantly faster than BDM. For the instances of 20 POIs, ILS superiority is particularly striking. Indeed, ILS outperforms BDM both in terms of solution profit and computing times. Overall, the ILS computed slightly worse solutions only for two instances. The average solution profit increase managed by the ILS method compared to the BDM is about 6.53% over all the test instances considered in Table 8.

Finally, Table 9 provides the results obtained by applying the proposed ILS method on larger BSP instances, involving up to 40 POIs and 60 groups, where again 5 ILS executions were made in order to attest its effectiveness over repeated runs. The same instance grouping and columns as in Table 8 are used. The results reported serve as a basis for future algorithmic comparisons.

Table 8
Comparison results on small-sized instances

				BDM		ILS				
$\|P\|$	$\|T\|$	VT	PT	*Profit*	*t* (seconds)	*Best profit*	*Avg profit*	*Best impr* (%)	*Avg impr* (%)	*t* (seconds)
10	30	1	PT-D	18,078	24.0	18,078	18,078	0.00	0.00	1.5
			PT-S	13,559	2748.3	13,565	13,565	0.04	0.04	3.5
			PT	14,736	2938.0	14,755	14,748.6	0.13	0.09	4.4
		2	PT-D	16,801	1.0	16,801	16,801	0.00	0.00	0.7
			PT-S	14,308	4.7	14,308	14,307.4	0.00	0.00	1.3
			PT	14,701	2.9	14,701	14,698.6	0.00	−0.02	1.2
		3	PT-D	18,024	4.7	18,028	18,028	0.02	0.02	1.1
			PT-S	14,398	104.3	14,398	14,398	0.00	0.00	2.3
			PT	15,189	71.2	15,189	15,172	0.00	−0.11	2.7
10	60	1	PT-D	34,097	4325.6	34,120	34,120	0.07	0.07	5.3
			PT-S	27,061	5534.1	27,108	27,102.2	0.17	0.15	12.7
			PT	28,312	6635.6	28,643	28,585.2	1.17	0.96	11.7
		2	PT-D	35,580	13.2	35,580	35,580	0.00	0.00	1.4
			PT-S	30,508	29.5	30,508	30,421.6	0.00	−0.28	2.6
			PT	29,758	234.6	29,758	29,724	0.00	−0.11	3.2
		3	PT-D	35,923	1462.9	35,908	35,848.2	−0.04	−0.21	4.5
			PT-S	30,225	4437.1	30,227	30,226.8	0.01	0.01	8.0
			PT	30,228	5478.4	30,287	30,196.8	0.20	−0.10	7.2
20	30	1	PT-D	11,023	7200.2	12,227	12,190	10.92	10.59	27.8
			PT-S	9712	7200.2	9842	9820	1.34	1.11	388.6
			PT	8425	7200.2	9047	8975.8	7.38	6.54	238.5
		2	PT-D	9776	3610.2	10,188	10,150.2	4.21	3.83	244.9
			PT-S	10,102	3626.7	10,262	10,206.8	1.58	1.04	340.6
			PT	8235	3628.4	8554	8425.2	3.87	2.31	307.6
		3	PT-D	10,947	4709.5	11,666	11,664.8	6.57	6.56	133.2
			PT-S	10,229	6151.3	10,298	10,257.2	0.67	0.28	353.2
			PT	8461	6675.5	8885	8805.4	5.01	4.07	295.9
20	60	1	PT-D	17,622	7200.4	24,220	24,185.4	37.44	37.25	116.8
			PT-S	11,473	7200.4	14,635	14,490	27.56	26.30	453.9
			PT	15,452	7200.4	16,893	16,700.4	9.33	8.08	352.4
		2	PT-D	20,844	6073.3	23,253	23,253	11.56	11.56	202.2
			PT-S	14,780	6659.3	15,744	15,642.8	6.52	5.84	430.4
			PT	15,721	7200.3	16,872	16,554.2	7.32	5.30	440.9
		3	PT-D	15,700	7200.3	24,891	24,776.6	58.54	57.81	58.4
			PT-S	9672	7200.3	15,669	15,612.4	62.00	61.42	436.9
			PT	11,095	7200.3	17,486	17,244.6	57.60	55.43	365.1
All				640,755	2933.2	682,594	680,556.2	6.53	6.21	146.2

Both BDM and ILS algorithms could be useful in supporting the decision-making in a sightseeing or travel company. As mentioned above, the BSP deals with a tactical problem faced by a company that wants to plan its bus tours and activities in anticipation. The designed bus routes can be part of *sightseeing packages* which are offered directly to tourists or sell to travel agencies as a proposal for touristic packages. According to the performance of BDM and ILS, the

Table 9
ILS results on instances with up to 40 POIs and 60 groups

$\|P\|$	$\|T\|$	VT	PT	*Best profit*	*Avg profit*	*t* (seconds)	$\|P\|$	$\|T\|$	VT	PT	*Best profit*	*Avg profit*	*t* (seconds)
30	30	1VT	PT-D	11,014	10,863.6	87.6	40	30	1VT	PT-D	7759	7559.2	101.6
			PT-S	8113	8016.8	475.1				PT-S	5326	5216.4	282.3
			PT	8066	7851.4	307.4				PT	5661	5523.4	214.1
		2VT	PT-D	9536	9372.8	338.8			2VT	PT-D	6789	6560.2	214.1
			PT-S	8357	8211.6	554.6				PT-S	5385	5253.4	416.4
			PT	7749	7622	481.7				PT	5503	5370	378.0
		3VT	PT-D	10,656	10,552	196.6			3VT	PT-D	7524	7292.4	116.9
			PT-S	8364	8280.2	509.0				PT-S	5442	5373.8	389.9
			PT	8101	7957.6	434.4				PT	5710	5638.6	274.8
30	60	1VT	PT-D	22,041	21,759.6	194.9	40	60	1VT	PT-D	15,616	15,314.2	212.0
			PT-S	15,955	15,674	471.4				PT-S	10,483	10,346.4	382.7
			PT	15,482	15,185.4	328.0				PT	10,380	10,160.8	316.8
		2VT	PT-D	21,214	21,105.2	507.6			2VT	PT-D	14731	14274.2	346.3
			PT-S	17,742	17,474.6	600.9				PT-S	11,400	11,095.2	590.2
			PT	15,783	15,376.8	584.6				PT	10,377	10,075.8	573.0
		3VT	PT-D	22,357	22,236.2	290.9			3VT	PT-D	15,610	15,297.8	257.3
			PT-S	17,503	17,275.6	547.7				PT-S	11,381	11,173.2	504.8
			PT	16,091	15,756.4	490.9				PT	10,673	10,399.6	422.9

computational times can be compatible with the operational requirements. For small-sized instances, BDM can find optimal solutions within a limited amount of computing time. For larger instances, the ILS is also applicable and its computing time can be tailored depending on the operational requirements. Also, in some situation where last-minute requests are encountered or some requirements are changed, a new solution can be efficiently obtained by performing local changes to an existing plan.

8. Conclusions and future research

In this paper, a new VRPP is introduced, called the BSP. The BSP calls for the determination of bus routes that maximize the collected profit enjoyed by tourists on board. To do so, decisions for two levels have to be jointly made: assignment of tourists to bus routes, routing of bus routes to POIs offered by the touristic destination considered. The BSP model generalizes the known challenging VRPP, such as the team OP with TW. In particular, it can be viewed as a generalization of the TTDP by considering multiple tours and different tourist preferences on the POI locations.

The paper proposes a BSP mathematical formulation. Based on this formulation, an exact logic-based Benders decomposition methodology is designed and presented. In addition, a metaheuristic algorithm based on an ILS framework is proposed, equipped with move types which solve MIP subproblems. The exact and heuristic algorithms are extensively assessed on a set of 576 newly constructed BSP benchmark instances. These test instances have been constructed to represent challenging BSP instances of diverse characteristics regarding several aspects of the problem, such as the size of POI and tourist group sets, composition of the bus fleet, duration of the sightseeing tours

and preferences of tourists to POIs. Several experiments aimed at testing the effectiveness of the various algorithmic components are performed and reported. The obtained results demonstrate that BSP instances with up to 20 POIs and 30 tourist groups can be consistently solved to optimality by the proposed exact method. In addition, high-quality solutions can be obtained by the proposed metaheuristic algorithm for instances involving up to 20 POIs and 60 groups and instances with up to 40 POIs and 60 tourist groups were solved by the metaheuristic algorithm.

We see two main topics for future research. First, we aim to investigate alternative mathematical formulations, from which deriving new dual bounds. In particular, we want to investigate set partitioning like formulations strengthened by adding valid inequalities and corresponding dual bounds computed in a column generation fashion. It is worth noting that, due to the structure of the BSP objective function, the classical set-partitioning formulation adopted to solve several variants of the vehicle routing problem (including problems with profits) cannot be used directly to model the BSP. Moreover, as pointed out also by Gavalas et al. (2014), TTDPs might have additional, complex operational requirements and constraints, such as accessibility features of sites, each bus can perform multiple routes during its working period, multiple TW, mandatory POIs preferences, to name a few. Hence, further work will be needed to investigate how these additional requirements can be embedded into our solution approaches. Furthermore, to further test the solution algorithms, future work will also focus on collecting data from real-world applications of the BSP.

Acknowledgments

We thank the associate editor and the anonymous reviewers whose constructive comments helped to improve the original version of this paper. The first two authors were financially supported by the National Natural Science Foundation of China (grant nos. 72101187, 72021002, and 72171111) and the Fundamental Research Funds for the Central Universities.

Open Access funding provided by the Qatar National Library.

References

Archetti, C., Speranza, M.G., Vigo, D., 2014. Vehicle routing problems with profits. In Toth, P., Vigo, D. (eds) *Vehicle Routing: Problems, Methods, and Applications*, Vol. 18. SIAM, Philadelphia, PA, pp. 273–297.

Baker, E., 1983. An exact algorithm for the time-constrained traveling salesman problem. *Operations Research* 31, 5, 938–945.

Baldacci, R., Mingozzi, A., Roberti, R., 2012. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* 24, 3, 356–371.

Baxter, J., 1981. Local optima avoidance in depot location. *The Journal of the Operational Research Society* 32, 9, 815–819.

Benders, J.F., 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4, 1, 238–252.

Codato, G., Fischetti, M., 2006. Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research* 54, 4, 756–766.

Crowder, H., Johnson, E.L., Padberg, M., 1983. Solving large-scale zero-one linear programming problems. *Operations Research* 31, 5, 803–834.

Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. *Transportation Science* 39, 2, 188–205.

Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., Linaza, M.T., 2013. Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research* 40, 3, 758–774.

Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., 2014. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics* 20, 3, 291–328.

Geoffrion, A.M., 1972. Generalized Benders decomposition. *Journal of Optimization Theory and Applications* 10, 4, 237–260.

Gunawan, A., Lau, H.C., Vansteenwegen, P., Lu, K., 2017. Well-tuned algorithms for the team orienteering problem with time windows. *Journal of the Operational Research Society* 68, 8, 861–876.

Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M., 2019. Variable neighborhood search. In Gendreau, M., Potvin, J.Y. (eds) *Handbook of Metaheuristics*. Springer International, Cham, pp. 57–97.

Hooker, J.N., 2000. *Logic-Based Methods for Optimization*. John Wiley & Sons, Hoboken, NJ.

Hooker, J.N., Ottosson, G., 2003. Logic-based Benders decomposition. *Mathematical Programming* 96, 1, 33–60.

Hu, Q., Lim, A., 2014. An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research* 232, 2, 276–286.

IBM CPLEX, 2018. IBM ILOG CPLEX 12.8.0 callable library. Available at https://www.ibm.com/support/pages/detailed-system-requirements-ibm-ilog-cplex-optimization-studio.

Labadie, N., Mansini, R., Melechovský, R. J. Wolfler Calvo, 2012. The team orienteering problem with time windows: an LP-based granular variable neighborhood search. *European Journal of Operational Research* 220, 1, 15–27.

Labadie, N., Melechovský, J., Wolfler Calvo, R., 2011. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics* 17, 6, 729–753.

Lourenço, H.R., Martin, O.C., Stützle, T., 2019. *Iterated Local Search: Framework and Applications*. Springer International, Cham. pp. 129–168.

Malucelli, F., Giovannini, A., Nonato, M., 2015. Designing single origin-destination itineraries for several classes of cycle-tourists. *Transportation Research Procedia* 10, 413–422.

Martello, S., Toth, P., 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research* 24, 11, 1097–1100.

Nemhauser, G.L., Wolsey, L.A., 1988. *Integer and Combinatorial Optimization*. Wiley Interscience, New York, NY.

Righini, G., Salani, M., 2006. Dynamic programming for the orienteering problem with time windows. Working Paper. University of Milan.

Ruiz-Meza, J., Montoya-Torres, J.R., 2022. A systematic literature review for the tourist trip design problem: extensions, solution techniques and future research lines. *Operations Research Perspectives* 9, 100228.

Savelsbergh, M.W.P., 1985. Local search in routing problems with time windows. *Annals of Operations Research* 4, 1, 285–305.

Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., Van Oudheusden, D., 2013. The multiconstraint team orienteering problem with multiple time windows. *Transportation Science* 47, 1, 53–63.

Vanderbeck, F., Wolsey, L.A., 2010. Reformulation and decomposition of integer programs. In Jünger, M., Liebling, T.M., Naddef, D., Nemhauser, G.L., Pulleyblank, W.R., Reinelt, G., Rinaldi, G., Wolsey, L.A. (eds) *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, Berlin, pp. 431–502.

Vansteenwegen, P., Gunawan, A., Vansteenwegen, P., Gunawan, A., 2019a. Applications of the OP. In *Orienteering Problems: Models and Algorithms for Vehicle Routing Problems with Profits*. Springer International, Cham, pp. 83–93.

Vansteenwegen, P., Gunawan, A., Vansteenwegen, P., Gunawan, A., 2019b. State-of-the-art solution techniques for OPTW and TOPTW. In *Orienteering Problems: Models and Algorithms for Vehicle Routing Problems with Profits*. Springer International, Cham, pp. 67–81.

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.

Wei, L., Zhang, Z., Lim, A., 2014. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine* 9, 4, 18–30.

Wei, L., Zhang, Z., Zhang, D., Lim, A., 2015. A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 243, 3, 798–814.

Zachariadis, E., Kiranoudis, C., 2011. A local search metaheuristic algorithm for the vehicle routing problem with simultaneous pick-ups and deliveries. *Expert Systems With Applications* 38, 2717–2726.

Zhang, Z., Luo, Z., Baldacci, R., Lim, A., 2021. A Benders decomposition approach for the multi-vehicle production routing problem with order-up-to-level policy. *Transportation Science* 55, 1, 160–178.