A Branch-and-Check Approach for the Tourist Trip Design Problem with Rich Constraints

Duc Minh Vu[a], Yannick Kergosien[b], Jorge E. Mendoza[c,*], Pierre Desport[b]

[a] *Cardiff Business School, Cardiff University, Aberconwy Building, Colum Drive, Cardiff, CF10 3EU*
[b] *Université de Tours, LIFAT EA 6300, CNRS, ROOT ERL CNRS 7002. 64 avenue Jean Portalis, 37200 Tours, France*
[c] *HEC Montréal. 3000 Chemin de la Côte-Sainte-Catherine, Montréal, QC H3T 2A7, Canada*

Abstract

The tourist trip design problem is an extension of the orienteering problem applied to tourism. The problem consists in selecting a subset of locations to visit from among a larger set while maximizing the benefit for the tourist. The benefit is given by the sum of the rewards collected at each location visited. We consider a variant of the problem that deals not only with "typical" constraints such as budget, opening-time hours (i.e., time windows at the locations), and maximum trip duration but also with other practical tourism constraints such as mandatory visits, limits on the number of locations of each type, and the order at which selected locations are visited. To solve this problem, we propose a branch-and-check approach in which the master problem selects a subset of locations, verifying all except time-related constraints, and these locations define candidate solutions to the master problem. For each candidate solution, the slave problem checks whether a feasible trip can be built using the given locations. To accelerate the branch-and-check approach, we propose and test improvements, including preprocessing to tighten the master-slave problem, valid inequalities generated dynamically to strengthen the master problem, and a local branching and variable neighborhood search to find new feasible solutions. Finally, we report the experimental results and compare the performance of the proposed exact algorithm with that of a mathematical solver.

Keywords: *tourist trip design*, *branch-and-check*, *exact method*

*Corresponding author
Email addresses: vudm@cardiff.ac.uk (Duc Minh Vu), yannick.kergosien@univ-tours.fr (Yannick Kergosien), jorge.mendoza@hec.ca (Jorge E. Mendoza), pierre.desport@univ-tours.fr (Pierre Desport)

1. Introduction

When visiting a city, tourists often ask what places they should visit and what the corresponding schedule should be. This is a challenging problem because large cities such as Paris, London, New York, or even medium-size cities such as Tours or Nice, offer numerous attractive locations to visit. Many cities now offer software and websites that help tourists plan their visits. References to such systems are given by Souffriau and Vansteenwegen (2010); Gavalas et al. (2014).

The present study stems from the "Smart Loire" project funded by the region of Tours (Centre Val de Loire) in an effort to aid tourists in planning their trip. The region has many attractive points of interest (POIs), such as castles, museums, and wine caves. Each specific location, for example, a castle, offers numerous possible activities for tourists, such as visiting the gardens, running through a maze, and eating at the castle restaurant. Therefore, deciding which locations to visit and which activities to undertake is not a simple question.

This problem can be seen as a tourist trip design problem, which is usually an extension of the orienteering problem (Vansteenwegen et al., 2011; Gunawan et al., 2016) with rich constraints. Most POIs require entrance fees and have opening times that constrain the times at which tourists can visit the sites. Given many POIs with different entrance fees, opening times, and other features to be detailed later, a tourist may want to plan a feasible schedule or trip that starts from a specific location (e.g., his hotel), visit a sequence of attractive locations, and finish his trip at a predefined location (e.g., his hotel or a new location). In this context, a tourist may envision partaking in many activities at each location. For example, when visiting a castle, he can either (1) visit only the castle, (2) visit the castle and its garden, (3) visit the castle and eat in a castle restaurant, or (4) simply visit the garden and eat in a restaurant. Therefore, each POI in this problem can be seen as a site and a set of activities associated with the site. To reflect the interest to tourists of a given POI, each POI is assigned a *score*, which can be adjusted based on the user profile to indicate the level of interest for each user in the given POI. A high score is indicative of an attractive location for the given user. The first step in solving this problem is to estimate the score of each POI, which may be done through surveys or by extracting historical travel patterns of visitors and historical tourism data of POIs (e.g., the annual number of visitors). We also divide POIs into categories (e.g., castle, museum, cathedral), where POIs can belong to several categories.

Besides common constraints such as the budget for entrance fees and the maximum duration of the trip, we consider additional aspects that reflect practical situations that tourists consider when designing their tour. The first such aspect is a short list of *mandatory* locations that the tourist must visit during his trip. Next, although the tourist is interested in visiting as many POIs as possible, he wants to avoid visiting too many POIs that have similar characteristics (e.g., modern art galleries). To implement this aspect, we enforce limits on the number of POIs that the tourist can visit in each category (e.g., gardens). For instance, a tourist can stipulate that he wants to visit at least one garden

and at most two. This type of constraint is called a *category* constraint. Next, a tourist region may have many historical sites (as is the case in the region of Tours), and the order in which they are visited can be important for tourists. Tourists usually prefer to visit such locations based on a timeline. For example, if we consider Honoré de Balzac, tourists want to visit first where he was born, then where he studied, where he lived after marriage, etc. The option of visiting according to a timeline is embodied in *precedence* rules. Also, some POIs may have more than shared stories or common features: they may have a logical order that would imply visiting a sequence of POIs in a certain order. This constraint is embodied in *implication* rules. Finally, the set of POIs corresponding to a single given site is expressed in an *exclusion* constraint, which means that at most one POI can be selected among the POIs corresponding to the given site.

Given a list of attractive locations with opening times, entrance fees, and a list of the tourist's personal preferences, it is not an easy task to devise, in a reasonable time, a trip that matches his expectations. Therefore, the Smart Loire project is striving to develop a system that offers hand-held devices to tourists to facilitate this step and help them enjoy their days in the Loire valley.

In this paper, we model this problem as an extension of the Orienteering Problem with rich constraints. To date, the few groups that have studied this problem include Souffriau et al. (2013), who consider time-window constraints and duration, budget, and category constraints, and Lin and Yu (2017), who consider mandatory visits and exclusion constraints. In our approach to this problem, we consider not only these constraints but also the practical constraints presented above. The goal of this work is to provide a method to solve this problem exactly because this will serve as a tool to evaluate the meta-heuristic methods to be proposed later in the Smart Loire project.

The paper is organized as follows: First, Section 2 summarizes the main articles related to the present research. Section 3 then presents the mathematical formulation, following which Section 4 presents the basic idea of branch-and-check and then details its customized algorithmic components. Section 5 presents the extensive experiments done to test and assess the proposed method and, finally, Section 6 concludes and gives directions for future research.

2. Literature review

The tourist trip design problem (TTDP) is a special variant of the orienteering problem (OP) and has recently attracted research attention because it has practical applications in logistics, tourism, scheduling, etc. For an exhaustive list of references to recent works on the OP and TTDP, please see the excellent reviews by Vansteenwegen et al. (2011); Gavalas et al. (2014); Gunawan et al. (2016). However, despite the numerous works that address these problems, few consider the question studied herein. To get an overview of such works, we now briefly review these papers.

First, note that most existing papers consider duration-related constraints and time-window constraints. Since 2010, mandatory constraints and exclu-

sion constraints have entered the discussion (see, e.g., Tricoire et al. (2010); Palomo-Martínez et al. (2017)). Tricoire et al. (2010) consider the problem in the context of planning individual routes for a software distribution company. The sales department needs to visit customers, and each customer may have at most two different time windows per day. The company has a list of important customers or mandatory customers that they visit on a regular basis; the remaining customers are considered optional. In addition to mandatory constraints, Tricoire et al. (2010) consider the duration constraint for each period and the total duration constraint for all periods. They finish by proposing a heuristic-based variable neighborhood search method for the problem.

Palomo-Martínez et al. (2017) proposed a greedy randomized adaptive search procedure (GRASP) and a variable neighborhood search (VNS) procedure to solve an orienteering problem with mandatory visits and exclusionary constraints. The problem emerges from the context of road maintenance: given a planning horizon, maintenance activities must be done, with some activities being mandatory to ensure traffic while other activities can wait for the next planning horizon. Some activities cannot be done on the same day because doing so could interrupt the traffic flow. The GRASP component is used to generate initial solutions. The route construction first includes mandatory vertices into a route and then tries to add optional vertices into a route. A VNS algorithm then explores the search space and identifies new solutions. The insert neighborhood operator tries to increase total profit, and the exchange neighborhood operator has the same goal by iteratively swapping the visited node with the lowest score with the unvisited node with the highest score. If the problem becomes unfeasible, a repair procedure is applied. Finally, the swap neighborhood operator changes the order of nodes in a route, with the goal being to reduce the length of the route. Within only a few seconds, the proposed algorithm can find better or equal solutions vis à vis CPLEX running up to 1 hour. In addition, the algorithm produces solutions within a 2% improvement for classic OP instances. In other work, Lu et al. (2018) proposed a memetic algorithm to solve the same problem. The proposed algorithm, which consists of tabu search and memetic operators (crossover, mutation), finds 104 new best solutions and obtains the best results for 236 instances.

Souffriau et al. (2013) studied a personal routing problem in the context of tourism by considering multiple constraints and multiple time windows. The so-called multi-constraint team orienteering problem with multiple time windows (MCTOPMTW) involves not only the regular duration constraint and time windows but also the budget constraint and max-n-type constraints. The max-n-type constraints impose the maximum number of locations of each type that can be visited (e.g., the maximum number of museums that can be visited on the first day). The authors present a hybrid greedy randomized adaptive search procedure and iterated local search to address the problem. First, they introduce the definition of neighborhood consistency, which helps accelerate the insertion and removal of a vertex. In addition, the authors define heuristic values that allow them to determine where to insert a new vertex. These components are used in a GRASP heuristic to generate solutions. The whole algorithm is an

iterated local search where the GRASP component is called each time a new solution needs to be generated. New (partial) candidate solutions are obtained by removing nodes from the current solution.

Lin and Vincent (2015) present a simulated annealing approach with restart strategy for the MCTOPMTW [Souffriau et al. (2013)]. Classic operators such as swap, insertion, and inversion are used to define neighbors given a candidate solution. The proposed algorithm obtains best solutions to 61 of the 148 benchmark instances and finds six new best solutions. Lin and Yu (2017) optimize a team orienteering problem with multiple time windows (TOPTW) with mandatory visits and total travel time and service time constraints. The algorithm is similar to the one mentioned by Lin and Vincent (2015).

To the best of our knowledge, whereas a few exact methods for the OP are mentioned in the above works, none of them studied the TTDP. As stated in the introduction, this paper not only considers existing constraints but also considers a new set of practical constraints in the domain of tourism. The exact algorithm presented herein is based on the branch-and-check approach, which was introduced by Thorsteinsson (2001).

3. Problem formulation

The problem is modeled as follows: A tourist starts his trip from a predefined initial location, visits selected POIs exactly once over predefined time windows, and then returns to a predefined final location. Let $V = \{0, \ldots, N\}$ be the set of POIs, where 0 and N are two special POIs denoting the starting and ending locations. With each POI we associate the following information: (1) opening time window $[e_i, l_i]$; (2) entrance fee b_i for the visit, (3) average duration t_i, and (4) the score p_i for POI i. Each POI may belong to several categories, and, given a set of categories C and a category $v \in C$, we define $V_v \subseteq V$ as the set of POIs that belongs to this category.

Let $A \subseteq V \times V$ be the set of travel arcs, and τ_{ij} for $(i, j) \in A$ be the travel time between POIs i and j. A feasible solution is a tour $\pi = (u_0 = 0, u_1, \ldots, u_m = N)$ that starts from 0, visits some POIs in V, terminates at N, and meets the following constraints:

1. Each selected POI is visited within its time window.
2. Neither the duration nor the budget of the tour exceeds D (units of time) or B (units of distance), respectively.
3. All POIs listed in the mandatory set M are visited.
4. The number of visited POIs of category $v \in C$ is between the lower bound lb_v and the upper bound ub_v.
5. The precedence rules $P \subseteq V \times V$, implication rules $I \subseteq V \times V$, exclusion rules $E \subseteq V \times V$ are satisfied. For each pair $(u, v) \in P$, u must be visited before v if both are visited. For each pair $(u, v) \in I$, v must be visited if u is visited. For each pair $(u, v) \in E$, either u or v can be visited, but not both.

To model the problem as a mixed integer linear program model, we use three sets of variables. The first type of binary variable $\{y_i\}_{i \in V}$ takes the value 1 (0) if POI i is (not) selected to be in the tour. The second type of binary variable, $x_{ij}\ \forall\ (i,j) \in A$, is 1 if a visit to site i is followed by a visit to site j, and 0 otherwise. The third type of variable, $s_i, i \in V$, gives the arrival time at location i.

Given the set V of POIs, TTDP(V) can be formulated as follows (where the constant $T_{ij} = l_i + t_i + \tau_{ij}$ serves as a big-M value):

$$\text{TTDP}(V): \ \max \sum_{i \in V} p_i y_i, \tag{1}$$

subject to

$$\sum_{i \in V} x_{0i} = \sum_{i \in V} x_{iN} = 1, \tag{2}$$

$$\sum_{(i,k) \in A} x_{ik} = \sum_{(k,j) \in A} x_{kj} = y_k\ \forall\ k \in V \backslash \{0, N\}, \tag{3}$$

$$s_i + t_i + \tau_{ij} - s_j \leq T_{ij}(1 - x_{ij})\ \forall\ (i,j) \in A, \tag{4}$$

$$y_i e_i \leq s_i \leq y_i l_i\ \forall\ i \in V, \tag{5}$$

$$\sum_{i \in V} b_i y_i \leq B, \tag{6}$$

$$s_N - s_0 \leq D, \tag{7}$$

$$s_i + (t_i + \tau_{ij}) y_i \leq s_j + T_{ij}(1 - y_j)\ \forall\ (i,j) \in P, \tag{8}$$

$$lb_\upsilon \leq \sum_{i \in V_\upsilon} y_i \leq ub_\upsilon\ \forall\ \upsilon \in C, \tag{9}$$

$$y_i + y_j \leq 1\ \forall\ (i,j) \in E, \tag{10}$$

$$y_i = 1\ \forall\ i \in M, \tag{11}$$

$$y_i \leq y_j\ \forall\ (i,j) \in \mathcal{Z}, \tag{12}$$

$$s_i \geq 0\ \forall\ i \in V, \tag{13}$$

$$y_i \in \{0,1\}\ \forall\ i \in V, \tag{14}$$

$$x_{ij} \in \{0,1\} \ \forall \ (i,j) \in A. \tag{15}$$

The objective function (1) maximizes the total profit of the tour. Constraint (2) ensures that the starting and ending locations are at site 0 and N, respectively. Constraint (3) ensures the connectivity of the tour. Constraint (4) enforces the arrival time relation between POIs i and j if they are visited consecutively. Constraint (5) means the visit can only happen when the site is open. Constraints (6) and (7) limit the total entrance fees and the duration of the visit. Constraints (8)–(12) are precedence constraints, category constraints, exclusion constraints, mandatory constraints, and implication constraints, respectively. Finally, constraints (13)–(15) define the domain of decision variables.

4. Solution method

To solve the TTDP, we propose a method that provides an exact solution and that is based on the branch-and-check framework (Thorsteinsson, 2001). The underlying idea of branch-and-check is to decompose the original problem into two smaller problems, called the *master* problem and the *slave* problem, both of which are easier to solve than the original problem. Given a solution to the master problem obtained while exploring its search tree, we check whether this solution is *feasible* to the slave problem. The design of the master and slave problems must ensure that a solution that is feasible for both the master and slave problems is also feasible for the original problem. This method has been used to solve several optimization problems such as the single machine scheduling problem (Sadykov, 2008), vehicle routing with time windows (Lam and Van Hentenryck, 2017), and wind-turbine maintenance scheduling (Froger et al., 2017).

Algorithm 1 explains the main steps of the basic version of the branch-and-check method. The master problem, which is denoted TTDP-M(V), aims to find the best subset of POIs that respects a subset of the constraints of the TTDP. When a solution s to the master problem is found while exploring the search tree, the method checks whether a feasible trip exists that visits all POIs selected in s. Generally, s corresponds to a *leaf* node of the search tree, but obtaining an integral solution s is possible while solving the relaxation at an internal node. The checking consists of solving a slave problem denoted TTDP-S(V_s). In any case, if a feasible trip exists [TTDP-S(V_s) is feasible], the method updates the best solution and adds an optimality to the master problem TTDP-M(V) to eliminate all but the best candidates. Otherwise, no feasible trip exists, and the method adds a cut to TTDP-M(V) to exclude all potential candidates trying to visit all POIs in s.

Algorithm 1: Basic Branch-and-Check Approach

Input: An instance of TTDP with rich constraints
Output: A feasible solution
1 **while** *not finished exploring the search tree of the master problem TTDP-M(V)* **do**
2 **if** *an integral solution s to the master problem is found* **then**
3 Check whether the slave problem TTDP-S(V_s) is feasible
4 Update the best solution if a new, improved solution is found
5 Tighten the master problem by using information while solving the slave problem.
6 Return the best solution found.

In addition to the branch-and-check, we propose a variant in which, instead of checking (line 3), we solve a small-size restricted TTDP problem defined on a subset $V_s \subseteq V$ and called TTDP-R(V_s), where V_s denotes the set of POIs belonging to solution s. The main idea is that, although it may be impossible to visit all POIs in s, a feasible and improved trip may be found by using a subset of those POIs. Thus, the TTDP-R(V_s) problem consists in solving the same TTDP problem (1)–(15) considering a small subset of POI. We name this variant *branch-and-solve*. Note that, if the optimal value returned by TTDP-R(V_s) is less than the value of s, then TTDP-S(V_s) is infeasible, and vice versa.

Since the number of candidates to check is exponential in the number of POIs, the basic framework cannot guarantee an acceptable performance for medium- and large-size instances. To improve the performance of the framework, we propose the following techniques:

- preprocessing to simplify input and tighten master and the slave formulations;
- cuts that are useful to tighten the master and the slave formulation;
- applying a method to generate a good initial solution and to find improved solutions;
- devising a strategy to identify feasible solutions and to stop branching on infeasible nodes while exploring the search tree of the master problem.

4.1. Formulations of master and slave problems

To solve the TTDP with branch-and-check, we define the formulations for the master and slave problems. Let TTDP-M(V) be the master formulation associated with the set V of POIs. Let TTDP-S(V_s) be the slave formulation, where s is a solution to TTDP-M(V). Here, V_s denotes a subset of V that only includes POIs belonging to the solution s.

Our master and slave formulations are the following:

Master problem

$$\text{TTDP-M(V)}: \ \max \sum_{i \in V} p_i y_i, \tag{16}$$

subject to

$$\sum_{i \in V} b_i y_i \leq B, \tag{17}$$

$$lb_v \leq \sum_{i \in V_v} y_i \leq ub_v \ \forall \ v \in C, \tag{18}$$

$$y_i + y_j \leq 1 \ \forall \ (i,j) \in E, \tag{19}$$

$$y_i = 1 \ \forall \ i \in M, \tag{20}$$

$$y_i \leq y_j \ \forall \ (i,j) \in I, \tag{21}$$

$$y_i \in \{0,1\} \ \forall \ i \in V. \tag{22}$$

Slave problem

Given a solution s to the master problem TTDP-M(V), we need to determine whether the slave problem has a feasible solution. We also denote as A_s, P_s the subsets of A, P considering only POIs belonging to the solution s. The slave problem TTDP-S(V_s) for a given s is defined as

$$\text{TTDP-S}(V_s): \ \min \ 0, \tag{23}$$

subject to

$$\sum_{i \in V_s} x_{0i} = \sum_{i \in V_s} x_{iN} = 1, \tag{24}$$

$$\sum_{(i,k) \in A_s} x_{ik} = \sum_{(k,j) \in A_s} x_{kj} = 1 \ \forall \ k \in V_s \backslash \{0, N\}, \tag{25}$$

$$s_i + t_i + \tau_{ij} - s_j \leq T_{ij}(1 - x_{ij}) \ \forall \ (i,j) \in A_s, \tag{26}$$

$$e_i \leq s_i \leq l_i \ \forall \ i \in V_s, \tag{27}$$

$$s_N - s_0 \leq D, \tag{28}$$

$$s_i + \tau_{ij} + t_i \leq s_j \ \forall \ (i,j) \in P_s, \tag{29}$$

$$x_{ij} \in \{0,1\} \ \forall \ (i,j) \in A_s. \tag{30}$$

When using the branch-and-solve approach, recall that, instead of solving TTDP-S(V_s) as above, we solve TTDP-R(V_s), which is defined by the same model of TTDP(V_s) without the upper-bound category constraint (9),

$\sum_{i \in V_v} y_i \leq ub_v \ \forall \ v \in C$, and the exclusion constraint (10) because s is a solution to the master problem.

4.2. Branch-and-check algorithm

Algorithm 2 highlights the main steps of the proposed approach. The algorithm starts with a preprocessing procedure that tightens time windows and eliminates unnecessary travel arcs. We also add valid inequalities to strengthen the master-slave formulations and other formulations used in the proposed algorithm (see Sections 4.2.1 and 4.2.2). Let *best* be the best integer solution found so far to the original problem; it initially takes the value 0 since no feasible solution is available. To examine potential candidate solutions to the master problem, we start with the set $\aleph$, which consists of unexplored nodes n. Each node n consists of fixed variables and unfixed variables. Fixed variables correspond to the variables y_i and take a value of either 1 (the tourist visits POI i in the current trip) or 0 (the tourist does not visit POI i in the current trip). Unfixed variables correspond to variables y_i that are not yet assigned a value.

Initially, $\aleph$ consists of a single node where all variables are unfixed. When a node n is selected to explore, we remove this node from $\aleph$, process it, and then decide whether we need to continue branching on this node if it contains unfixed variables. When branching, we branch on an unfixed variable y_i of n and we add at most two new nodes n_1 (corresponding to y_i fixed at 0) and n_2 (corresponding to y_i fixed at 1) to $\aleph$ while holding the other variables in n unchanged. We add n_1 (n_2) to $\aleph$ only if its relaxation value is better than the value of *best*. Since the number of unfixed variables is bounded, the algorithm should eventually terminate. When *best* is updated, we also remove from $\aleph$ all nodes whose optimal relaxation value is not greater than *best*.

Lines 4–28 of Algorithm 2 show how our branch-and-check algorithm works. While exploring the search tree, two main cases are considered when exploring node n. The first case is when a candidate solution is found at the current node n. This situation occurs when either (1) all variables are fixed (a leaf node) or (2) the solution to the continuous relaxation of TTDP-M(V) is actually integer [e.g., unfixed variables of this node take integer values in an optimal solution to the linear relaxation of TTDP-M(V)]. The second case happens when n is *an internal node*, we check whether we should continue branching at this node or stop branching due to the violations of the time-window constraints. We now explain both cases.

Processing an integer candidate solution to the master problem. Lines 6–12 of Algorithm 2 show the steps corresponding to the first case. Given a feasible solution s to TTDP-M(V) found at node n, we check whether s depicts a feasible solution to the original problem. Before solving TTDP-S(V_s), we apply TSPTW preprocessing steps (tighten time windows, eliminate redundant arcs, see Dumas et al. (1995); Vu et al. (2019) for the details) since the slave problem is a variant of TSPTW. If TTDP-S(V_s) is feasible, then we obtain a feasible solution to the original problem TTDP(V). In this case, we call a local branching and a variable neighborhood search algorithm called LB + VNS(s) (see Section 4.2.3) to find new, improved solutions. Since a feasible solution is

Algorithm 2: Branch-and-Check framework

Input: An instance of TTDP with rich constraints
Output: The best solution found

```
1  Call preprocessing procedure
2  Let best ← ∅ be the current best feasible solution
3  Let ℵ be a set of unexplored nodes that is initialized with a node where
     all variables are unfixed
4  while ℵ ≠ ∅ do
5      Select a node n ∈ ℵ to process:
6      if the optimal solution to the relaxation problem associated to n is
         integer then
7          if [TTDP-S(V_s) is feasible] then
8              Call LB + VNS(s) procedure to find new, improved solutions
9              Add cut (31) to the master problem
10             Stop branching on this node
11         else
12             Add cuts (33) to the the master problem: Algorithm 3;
13     if n is an internal node then
14         Get the partial solution s̄ - set of POIs that are planned to visit
15         if (best = ∅) then
16             Call LB + VNS(s̄) procedure to find feasible solutions
17         if [v(s̄) ≤ v(best) and |V_s̄| ≤ l_feasible] or (lower-bound category
             constraint is violated) then
18             Branch on an unfixed variable of n and update ℵ
19         else
20             if [TTDP-S(V_s̄) is infeasible] then
21                 Add cuts (33) to the master problem
22                 Stop branching on this node
23             else
24                 if [v(s̄) ≥ v(best)] then
25                     Call LB + VNS(s̄) procedure to find new, improved
                         solutions
26                     Add cut (31) to the master problem
27                 Branch on an unfixed variable of n and update ℵ
28 return best
```

found at this node, we stop branching at this node and add the cut (31) to the master problem to prune all nodes and candidate solutions which are not better than the current best solution. This cut is implemented whenever a new best solution is found. Here, $v(s)$ gives the value of solution s, so $v(best)$ is the best lower-bound value found:

$$\sum_{i \in V} p_i y_i \geq v(best) + 1. \tag{31}$$

If no feasible solution is found (line 12), it means that those POIs cannot be visited together. The candidates that visit all POIs in V_s are then removed by using the valid inequality (32) or its strengthened form (33). The notation $|V_s|$ in the following valid inequality denotes the number of POIs belonging to solution s:

$$\sum_{i \in V_s} y_i \leq |V_s| - 1. \tag{32}$$

The cut (32) means that the tourist can visit at most $|V_s| - 1$ POIs in s. Note that, since the time-window constraints are not considered in the master formulation, the number of POIs that can be visited together in V_s is generally much smaller than $|V_s| - 1$. Therefore, we determine an upper bound of this number, which is denoted TTDP-MAX-VISIT(V_s), and we propose a stronger valid inequality (33):

$$\sum_{i \in V_s} y_i \leq \text{TTDP-MAX-VISIT}(V_s). \tag{33}$$

The value TTDP-MAX-VISIT(V_s) is obtained by solving

$$\text{TTDP-MAX-VISIT}(V_s): \ \max \sum_{i \in V_s} y_i,$$

with respect to all TTDP constraints (2)–(15) except category constraint (9) and exclusion constraint (10). We ignore the exclusion constraint (10) since s is a solution to TTDP-M(V), which is also a reason why we ignore upper-bound category constraints. We ignore lower-bound category constraints because, given a solution to TTDP-MAX-VISIT(V_s), we may possibly add POIs belonging to V and V_s to obtain a feasible solution to TTDP-M(V). This valid inequality is proven experimentally to be effective because the value of TTDP-MAX-VISIT(V_s) is much less than $|V_s| - 1$.

Note that the master formulation is a weak relaxation of the original formulation (no time-window constraints), so the number of POIs in a candidate solution s may be much larger than the number of POIs in the feasible solutions. In this case, solving TTDP-MAX-VISIT(V_s) is time consuming, and TTDP-MAX-VISIT(V_s) and V_s can be reduced to make constraint (33) more effective. Algorithm 3 identifies a small subset of V_s that violates the time-window constraints and provides another opportunity to find new feasible solutions. Al-

gorithm 3 orders the POIs in s by placing mandatory POIs at the beginning and then sorting the remaining POIs decreasingly by their score. Scores are sorted in decreasing order because this approach matches our branching strategy. Next, we select a subset of POIs such that the total score is better than the current best solution. If we cannot visit all POIs of this subset, we add a cut to the master model and stop the procedure. Otherwise, we seek improved solutions by calling the local branching and variable neighborhood search procedure and then continue our steps until a violated subset is found. Algorithm 3 terminates because we know that the input of the algorithm is a solution s in which TTDP-S(V_s) is infeasible.

Algorithm 3: Determination of violated sets

Input: A set of POIs $\{u_1, u_2, \ldots, u_m\}$ which constitutes an infeasible solution s to TTDP-M(V)

Output: A subset of POIs that violates time-window constraints

1 Order the POIs in s such that
2 - Mandatory POIs are placed at the beginning in the resulting list
3 - Other POIs are sorted decreasingly by score
4 Determine the smallest value of k so that $\sum_{1 \le i \le k} p_{u'_i} \ge v(best)$
5 Solve TTDP-MAX-VISIT($\{u'_1, \ldots, u'_k\}$)
6 **if** *TTDP-MAX-VISIT*($\{u'_1, \ldots, u'_k\}$) $\le k - 1$ **then**
7 Add the corresponding cut (33) to TTDP-M(V) considering that $V_s = \{u'_1, \ldots, u'_k\}$
8 **else**
9 Call LB + VNS($\{u'_1, \ldots, u'_k\}$) to find improved solutions
10 Let $k \leftarrow k + 1$ and return to Line 4 of this algorithm

Processing an internal node. When n is an internal node (i.e., the second case, lines 14–27), we want to determine whether branching on unfixed variables of n is interesting. We start by identifying POIs that are assumed to be visited (line 14, denoted $\bar{s}$). These POIs actually correspond to the variables y_i, which are fixed to be 1 in the current node n. First, if no feasible initial solution is found, a call to LB + VNS($\bar{s}$) is made to obtain, as soon as possible, an initial solution (line 16). The main idea of lines 17–27 is to prune the search tree as soon as possible. When it is impossible to visit all POIs in $\bar{s}$ (lines 20–22), we add a cut (33) to the master problem and stop branching on this node. Otherwise, if all POIs in $\bar{s}$ can be visited, we call LB + VNS($\bar{s}$) to find improved solutions only if $\bar{s}$ is at least as large as the current best solution. Since all these steps (lines 20–27) can be time consuming, we check the feasibility of the slave problem only when the total score of POIs that are planned to visit is at least as large as the current best [$v(\bar{s}) > v(best)$] or when the number of POIs in $\bar{s}$ is sufficient ($|V_{\bar{s}}| > l_{\text{feasible}}$). Candidate solutions also need to satisfy lower-bound category constraints. The quantity l_{feasible} estimates an upper bound for the number of POIs in any feasible trip, and this value is updated dynamically as follows:

- Initially, $\alpha = 0$ and $\beta = 0$, where α, β denote the total of the length of feasible trips and the number of trips that we have checked, respectively.
- After each call to TTDP-MAX-VISIT(V_s) (or when the best solution *best* is updated):
  - $\alpha \leftarrow \alpha +$ TTDP-MAX-VISIT(V_s) (or $+|V_{best}|$),
  - $\beta \leftarrow \beta + 1$.
- $l_{\text{feasible}} \leftarrow \lceil \frac{\alpha}{\beta} \rceil$.

Although TTDP-MAX-VISIT(V_s) can be solved efficiently in most cases (e.g., from milliseconds up to a few seconds during the experimentation), it does not need to be solved optimally. If the resolution time exceeds 15 s, we replace TTDP-MAX-VISIT(V_s) by its upper bound, which is $\min(|V_s| - 1, \lfloor \text{UB(TTDP-MAX-VISIT}(V_s)) \rfloor)$, where UB(TTDP-MAX-VISIT(V_s)) returns an upper bound while solving TTDP-MAX-VISIT(V_s).

While exploring the search tree, many options may be available from which to select the node $n \in \aleph$ to be processed. In the proposed algorithm, we process node n with the largest relaxation value, and we branch on an unfixed variable that is active in the current node (e.g., $y_i > 0$) and that has the largest score p_i. Experimentation shows that such a strategy is effective and helps to reduce the upper bound more than other strategies.

Finally, the branch-and-solve variant of the proposed branch-and-check mentioned in Algorithm 2 is obtained by replacing the condition "TTDP-S(V_s) is feasible" in line 7 by "v(TTDP-R(V_s)) = $v(s)$," because we know that, if TTDP-S(V_s) is feasible, then TTDP-R(V_s) = $v(s)$. We also add constraint (31) to TTDP-R(V_s) before solving this problem because we are only interested in solutions that are better than the current best solution while solving the restricted problem. This valid inequality also helps to accelerate the resolution of TTDP-R(V_s) because it tightens the search space of TTDP-R(V_s).

In the following sections, we discuss in more detail the preprocessing, valid inequalities, and local branching and variable neighborhood search components of our algorithm.

4.2.1. Preprocessing

As mentioned previously, preprocessing should be done to tighten the master and the slave formulation. In our problem, a series of steps is considered, including (1) TSPTW-like preprocessing to eliminate infeasible arcs and tighten time windows [see, e.g., Dumas et al. (1995)], (2) deduction from the implication set, and (3) deduction from the exclusion set.

If $x \rightarrow y$ is an implication, the following rules are used to refine the implication set:

- If x is a mandatory POI, then y may be considered a mandatory POI.
- If y is a mandatory POI, we remove this rule from the implication set.

After applying these rules, the implication set includes only rules where all associated POIs are optional POIs.

Let $\text{imp}(x) \subseteq V$ be the set of POIs that the tourist visits when he visits POI x. For example, if the implication set includes rules $\{x \to y,\, x \to z,\, y \to r\}$, then $\text{imp}(x) = \{x, y, z, r\}$. Actually, $\text{imp}(x)$ can be built by using a traversal method in graph theory. The following observations are used to refine the exclusion set E:

- If $(u, v) \in E$ and $u, v \in M$, then E is an invalid instance.
- If $(u, v) \in E$ and $u \in M$, we remove v from the set of POIs.
- If the visit of all POIs in $\text{imp}(x) \cup \text{imp}(y)$ violates budget or category constraints, then (x, y) is an exclusion pair.
- If $\exists\ u \in \text{imp}(x)$ and $\exists\ v \in \text{imp}(y)$ where $(u, v) \in E$, then (x, y) is also an exclusion pair.

4.2.2. Valid inequalities to strengthen master and slave problem

We exploit time-window constraints and duration constraints to establish several valid inequalities to tighten the master problem. First, if we cannot visit both POIs i and j in the same trip because the time-window constraint is violated (i.e., $e_i + \tau_{ij} + t_i > l_j$ and $e_j + \tau_{ji} + t_j > l_i$), we have

$$y_i + y_j \leq 1\ \forall\ i, j \in V, \quad e_i + \tau_{ij} + t_i > l_j, \quad e_j + \tau_{ji} + t_j > l_i. \tag{34}$$

Next, we develop valid inequalities related to the duration constraint. If we cannot visit both POIs i and j because of the duration constraint, $l_i + D < e_j$ or $l_j + D < e_i$, then we have

$$y_i + y_j \leq 1\ \forall\ i, j \in V, \quad \min(l_i, l_j) + D < \max(e_i, e_j). \tag{35}$$

The duration constraint also ensures that the total travel time and visit duration must be less than D. Therefore, we have

$$\sum_{i \in V \setminus N} \left(\underline{\tau}_i^{\text{out}} + t_i\right) y_i \leq D, \tag{36}$$

$$\sum_{i \in V \setminus 0} \left(\underline{\tau}_i^{\text{in}} + t_i\right) y_i \leq D, \tag{37}$$

where $\underline{\tau}_i^{\text{out}} = \min_{(i,j) \in A} \tau_{ij}$ and $\underline{\tau}_i^{\text{in}} = \min_{(j,i) \in A} \tau_{ji}$ denote the minimum outbound travel time and minimum inbound travel time for a given POI i, respectively.

To tighten the slave formulation TTDP-S(V_s), we rely on another relaxation of the duration constraint. These constraints state that the total travel time

and visiting time should not exceed the duration of the visit:

$$\sum_{(i,j)\in A} x_{ij}(\tau_{ij} + t_i) \leq D, \tag{38}$$

$$\sum_{(i,j)\in A} x_{ij}(\tau_{ij} + t_j) \leq D. \tag{39}$$

We exploit the valid inequality (33) by solving exactly a variant of TTDP-MAX-VISIT(V) but with all TTDP constraints in a maximum of 60 s and adding a cut of the form $\sum_{i\in V} y_i \leq$ TTDP-MAX-VISIT(V) to TTDP-M(V) to limit the maximum number of POIs in any candidate solution of the master problem. This approach also helps tighten the search space of the master problem and improve overall performance.

4.2.3. Local branching and variable neighborhood search

The LB + VNS(V_s) procedure is based on the ideas of local branching (Fischetti and Lodi, 2003) and variable neighborhood search (Mladenovic and Hansen, 1997), which are two well-known heuristic approaches to search for new, improved solutions. Given a solution s and a parameter d, we define a neighborhood $N_s = \{v \in V \mid \exists\ u \in V_s \text{ such that } \tau_{uv} \leq d\}$ of s, which consists of POIs in V that either belong to s or are close to at least one POI in s. Here, $\tau_{uv} \leq d$ indicates that the distance between u and v does not exceed d. Since N_s consists of POIs close to V_s, it is possible to insert some POIs from $N_s \backslash V_s$ into the current solution or swap POIs between $N_s \backslash V_s$ and V_s to obtain new solutions. We solve TTDP-LB(N_s, k, d), which is the model of TTDP(N_s) plus the local branching constraint (40). This constraint forces the total number of POIs that we can either remove from s and/or insert into s to be at most k:

$$\sum_{i\in V_s} (1 - y_i) + \sum_{i\in N_s\backslash V_s} y_i \leq k. \tag{40}$$

The algorithm maintains a data structure named m in which $m(s)$ gives how many times TTDP-LB(N_s, k, d) has been solved for a particular solution s. If TTDP-LB(N_s, k, d) has yet to be solved for any pair (k, d), $m(s)$ returns 0.

Algorithm 4: Local Branching and Variable Neighborhood Search: LB + VNS(s)

Input: A (feasible) solution s to TTDP(V)
Output: A (possible) new, improved solution

```
1  m(s) ← m(s) + 1;
2  if m(s) > |Ω|, stop;
3  (k, d) ← the m(s)th (k, d) pair in Ω—the set of all (k, d) pairs;
4  while i ≤ |Ω| do
5      Define the neighborhood N_s of s based on the value d;
6      Solve TTDP-LB(N_s, k, d) with the cut (40) for the parameter k;
7      if a new, improved solution s' is found then
8          Update s ← s';
9          m(s) ← 1;
10         (k, d) ← the 1st (k, d) pair in Ω;
11     else
12         m(s) ← m(s) + 1;
13         (k, d) ← the m(s)th (k, d) pair in Ω;
14 return s;
```

In our implementation detailed in Algorithm 4, we maintain a list of (k, d) pairs in an ordered list Ω, and each element in Ω is used at most once. For each input s, we determine the number of times we call TTDP-LB(N_s, k, d) for some pairs of (k, d) (line 1). This can be done by using a hashing technique. We hash the POIs in s into an integer number, and we store the number of times TTDP-LB(N_s, k, d) has been called for each s. If all pairs in Ω are used, we stop (line 2). Otherwise, we define the neighborhood N_s and solve the corresponding TTDP-LB(N_s, k, d). If a new, improved solution s' is found, we reset (k, d) to the first pair in Ω. Otherwise, we adjust (k, d) to the next pair in Ω.

5. Experiments

This section assesses the performance of the proposed algorithm when solving TTDP. The algorithm was implemented in C++, and all experiments were run on a workstation with an Intel(R) Xeon (R) CPU E5-4610 v2 2.30 GHz processor running the Ubutu Linux 14.04.3 Operating System. We used CPLEX Concert 12.8 API to formulate and solve all MIP formulations. We relied on CPLEX's callback library to access integer solutions and internal nodes.

All parameter values for CPLEX are left at their default values except for (1) the number of threads, *Threads*, which is set to 1 and (2) the heuristic strategy, *HeuristicFreq*, which is disabled when solving TTDP-M(V). The stopping conditions are (1) a one hour time limit or (2) when an optimal solution is found. In our case, since the profits p_i are integers, the sum $\sum_{i \in V} y_i p_i$ is also an integer. Therefore, optimality is reached when the difference between the upper and lower bounds is less than 1. We implemented the checking steps, valid inequalities (31)–(33), node selection, and branching strategy as callbacks.

When we do not activate the proposed node selection and own branching strategy (e.g., in our basic branch-and-check version—Algorithm 1), we let CPLEX decide which nodes to process. Regarding the local branching and variable neighborhood search procedure, Ω includes three pairs in our implementation, $\{(10, 15), (15, 10), (20, 5)\}$. We solve TTDP-LB($N_s, k, d$) for at most 60 s for each pair.

We tested our algorithms on the instances extended from the MCTOPMTW (called SC-O; see Table 1), proposed by Souffriau et al. (2013). Briefly, the original set consists of 37 instances, of which 29 are extended from the well-known Solomon dataset (Solomon, 1987), and the remaining 8 extended from instances proposed by Cordeau et al. (1997). The initial and destination locations are assumed to be the same. The duration constraint is assumed to be the duration of the time window at the departure location. The authors extend existing instance sets to include the budget constraint and ten upper-bound category constraints. To model the budget constraint, each POI is given a random entrance fee between 0 and 99. To model the category constraint, each POI i has a 0.2 probability of belonging to category c for each $c \in C$. Therefore, each POI may belong to several or zero categories. Each category has an average of 22 POIs and 12 POIs belong to no category. We consider the category constraints in that paper Souffriau et al. (2013) as our upper-bound constraints. The lower bound constraints are generated by using the rules presented below. Since Souffriau et al. (2013) consider multiple time windows whereas our problem assumes a single time window, the time window for each POI in the benchmark instance is defined by the time interval [opening of the first time window, closing of the last time window] of the corresponding POI of Souffriau et al. (2013).

	No. Instances	No. POIs	No. Categories
Solomon et al.	29	100	10
Cordeau et al.	8	48–233	10

Table 1: Characteristics of original benchmark instances (Souffriau et al., 2013).

We now explain how to generate new constraints. Let $s = (u_0, u_1, u_2, \ldots, u_m)$ and its associated POI set $V_s = \{u_0, u_1, u_2, \ldots, u_m\}$ depict a solution to the MCTOPMTW obtained after solving the problem by applying CPLEX for 10 hours. The idea is to ensure that s remains a feasible solution to the new instances corresponding to the problem being studied. First, the mandatory POIs are POIs randomly selected in V_s. In our experiment, the number of mandatory POIs is a random integer in the interval $[2, 5]$. Second, to generate lower-bound constraint, we first compute nb_c, which is the number of POIs of type c that is in s. Next, lb_c is set to a random number in the interval $[0, nb_c]$. To generate precedence constraints, implication constraints, and exclusion constraints, we follow the rules given below. The number of constraints of each

type is set to 1% of the number of possible constraints of each type. Given a random ordered pair (p, q) of $V \times V$ POIs, we apply the following constraints:

- Implication constraint: (p, q) can be in I if either (1) $p, q \in V_s$, or (2) $p \in V \backslash V_s,\ q \in V$.
- Exclusion constraint: (p, q) can be in E if at least p or q is in $V \backslash V_s$.
- Precedence constraint: (p, q) can be in P if (1) $p = u_i \in V_s,\ q = u_j \in V_s$ with $i < j$, or (2) $p \in V \backslash V_s,\ q \in V$.

This strategy ensures that s remains a feasible solution to the new benchmark instances. Table 2 presents the notation of the benchmark instances. SC-M means that mandatory constraints are considered (or $M \neq \emptyset$), SC-MLB means that mandatory constraints and lower-bound category constraints are considered, etc. Instances called SC-MLBICP are instances of the TTDP with all constraints applied.

Legend	Constraints
M	Mandatory constraints
LB	Lower-bound category constraints
I	Implication constraints
C	Exclusion constraints
P	Precedence constraints

Table 2: Notation for benchmark instances.

We report in Section 5.1 the results returned by the MIP solver, CPLEX 1h (i.e., within the time limit of 1 hour) and CPLEX 5h (i.e., within the time limit of 5 hours), as well as results returned by the basic branch-and-check and basic branch-and-solve, and those returned by the proposed branch-and-check and branch-and-solve. We report in Section 5.2 the contributions of each component of the proposed branch-and-check and branch-and-solve.

5.1. Overall performance

Table 3 explains the notation used in the following tables. We report the number of instances for which optimal solutions are found (No. Optimal), the number of instances where feasible solutions are found (No. Fea. Sol.), the number of times candidate solutions s are checked (No. Check), and the number of times an internal node is pruned while branching (No. Reject).

First, we report in Table 4 the results obtained by CPLEX after 1 hour and 5 hours of execution. The results show that, in general, the TTDP with rich constraints is hard to solve optimally. Although CPLEX optimally solves all of the original instances of Solomon when we only consider the first 50 POIs, CPLEX fails to solve most of them when all POIs are considered. In addition, CPLEX fails to find feasible solutions to several instances when lower-bound

Legend	Meaning
No. Optimal	No. of instances that are optimally solved
No. Fea. Sol.	No. of instances where at least one feasible solution is found
No. Reject	No. of nodes pruned while branching
No. Check	No. of slave or restricted problems solved
LB	Lower bound
UB	Upper bound
Improvement (%)	Average improvement: $100\% \times (\text{UB} - \text{LB})/\text{UB}$ of unsolved instances
B-BaC or P-BaC	Basic or proposed branch-and-check algorithm
B-BaS or P-BaS	Basic or proposed branch-and-solve algorithm
A vs B (%)	Improvement is calculated as $100\% \times (B - \text{A})/B$

Table 3: Notation.

constraints are considered. We report in the last two rows of Table 4 the percent improvement in the lower and the upper bounds obtained between 1 hour and 5 hours, respectively. These results show that, although CPLEX obtained improved lower bounds, it has difficulty reducing the upper bounds. As a result, only a few more instances are optimally solved.

		SC-O	SC-M	SC-MLB	SC-MLBI	SC-MLBIC	SC-MLBICP
1h	No. Optimal	5	7	9	11	11	11
	No. Fea. Sol	37	37	28	33	31	34
	Improvement (%)	40.25	41.47	58.65	44.38	47.65	39.74
5h	No. Optimal	7	8	9	11	12	11
	No. Fea.Sol	37	37	28	33	32	34
	Improvement (%)	37.77	38.81	54.65	42.92	45.00	37.90
1h vs 5h	LB improvement (%)	5.19	3.52	2.22	1.00	3.26	1.11
	UB improvement (%)	−1.06	−1.70	−0.70	−0.50	−1.09	−0.76

Table 4: Statistics for results returned by CPLEX after execution for 1 hour and 5 hours.

Table 5 reports the results obtained by the basic branch-and-check and branch-and-solve approach (Algorithm 4.2.1), where the valid inequalities (38) and (39) have been added to the slave formulations TTDP-S(V_s) and TTDP-R(V_s), respectively. The results obtained by the basic approach are inferior to those returned by CPLEX. Note also that, although numerous candidate solutions are checked, the basic branch-and-check approach fails to find feasible solutions to many instances. One reason for this result is that the master formulation is a weak relaxation of the original problem, so the candidate solutions usually violate time-related constraints. This indicates the need for a feasible method to generate solutions to improve the performance of the approaches studied. A possible way to solve the feasibility issue is to apply the branch-and-solve approach. By doing so, we obtain feasible solutions for all 37 instances of each set. However, this does not suffice to improve the performance of the basic

approach in terms of the number of optimal solutions. We need more time to solve an instance of TTDP-R(V_s) than an instance of TTDP-S(V_s). Therefore, fewer candidates are checked when applying the branch-and-solve method, so the upper bound returned by branch-and-solve is inferior to the upper bound returned by branch-and-check.

		SC-O	SC-M	SC-MLB	SC-MLBI	SC-MLBIC	SC-MLBICP
B-BaC	No. Optimal	0	4	5	9	10	10
	No. Fea. Sol.	6	8	9	17	21	21
	Improvement (%)	88.36	90.04	90.01	75.72	66.58	66.65
	No. Check	61 918	68 957	63 200	62 329	60 785	60 856
B-BaS	No. Optimal	0	4	6	8	9	9
	No. Fea.Sol	37	37	37	37	37	37
	Improvement (%)	41.12	41.10	42.08	33.90	32.56	32.61
	No. Check	18 260	14 131	12 632	18 140	18 372	18 950

Table 5: Results of basic branch-and-check and branch-and-solve approach.

Table 6 gives the percent improvement between the upper bound obtained by CPLEX and those obtained by the basic branch-and-check and branch-and-solve methods. Since the master TTDP-M(V) is easy to solve, many candidate solutions are checked and many simple cuts (32) are added to the master formulation. Except for the set SC-O, the upper bound returned by the basic branch-and-check is better on average than the upper bound returned by CPLEX (this is indicated by a negative percent "improvement"). Recall that the improvement is calculated as $100\% \times (\text{UB B-BaC} - \text{UB CPLEX})/\text{UB B-BaC}$ or $100\% \times (\text{UB B-BaS} - \text{UB CPLEX})/\text{UB B-BaS}$ for the branch-and-check and branch-and-solve methods, respectively. The basic branch-and-solve method produces inferior upper bounds because much fewer solutions are checked than with the branch-and-check method. In any case, improving the performance of the basic branch-and-check and branch-and-solve methods by using customized components should significantly help to improve the performance of the basic approaches.

	SC-O	SC-M	SC-MLB	SC-MLBI	SC-MLBIC	SC-MLBICP
UB CPLEX vs UB B-BaC (%)	2.15	−0.66	−0.37	−4.31	−4.85	−4.34
UB CPLEX vs UB B-BaS (%)	3.52	1.85	1.72	−0.48	−1.28	−0.79

Table 6: Percent difference between upper-bound values returned by CPLEX 1h and those return by the basic branch-and-check (BaC) and branch-and-solve (BaS) approaches.

Table 7 shows the results of the proposed branch-and-check and branch-and-solve approach (Algorithm 2) with a time limit of 1 hour. When the problem is more constrained, the number of solved instances increases, as seen in the row "No. Optimal." This result is attributed to the solution space of the master problem being much smaller when the problem is more constrained. We solve

30 of the 37 instances of the SC-MLBICP benchmark set (which is our TTDP), whereas we solve only 20 of the 37 instances of the original set SC-O. The more constraints a problem has, the more effective the approach is because the search space of the master problem is much more strongly bounded, resulting in significantly fewer candidate solutions to check. These results indicate that the proposed approach thoroughly outperforms CPLEX. Also, the values of "No. Check" and "No. Reject" show that much fewer TTDP-S(V_s) or TTDP-R(V_s) need to be solved than for the basic branch-and-check and branch-and-solve, which indicates the strength of the proposed cuts and branching strategy in terms of eliminating invalid candidate solutions and invalid partial solutions. The average improvement between unsolved instances remains large mainly because of unsolved instances of the Cordeau class. The rows "LB vs LB CPLEX 5h" and "UB vs UB CPLEX 5h" show the improvement in our lower and upper bounds compared with those obtained by CPLEX 5h. The percent improvement is calculated as by $100\% \times$ (LB P-BaC $-$ LB CPLEX 5h)/LB P-BaC and $100\% \times$ (UB P-BaC $-$ UB CPLEX 5h)/UB P-BaC, for the P-BaC, and similarly for P-BaS. Positive (negative) improvements for the lower (upper) bound indicate that P-BaC returns a better lower (upper) bound than CPLEX. The improved lower and upper bounds returned by the proposed algorithm versus CPLEX indicates the effectiveness of the proposed components, including local branching, branching strategies, and valid inequalities. Finally, note that branch-and-check performs slightly better than branch-and-solve, which we attribute to the longer time required to solve TTDP-R(V_s) than to solve TTDP-R(V_s), so fewer candidate solutions and nodes are explored by branch-and-solve than by branch-and-check.

		SC-O	SC-M	SC-MLB	SC-MLBI	SC-MLBIC	SC-MLBICP
P-BaC	No. Optimal	17	23	22	28	30	30
	No. Fea. Sol.	37	37	37	37	37	37
	Improvement (%)	22.48	25.91	24.81	29.31	33.13	33.12
	No. Check	441	369	396	226	212	193
	No. Reject	4458	1727	1562	724	897	760
	LB vs LB CPLEX 5h (%)	3.09	2.14	25.46	12.07	14.95	9.31
	UB vs UB CPLEX 5h (%)	−25.32	−31.92	−28.61	−24.81	−23.39	−23.05
P-BaS	No. Optimal	17	22	22	28	29	30
	No. Fea. Sol.	37	37	37	37	37	37
	Improvement (%)	23.65	23.04	26.69	29.84	31.14	37.44
	No. Check	263	240	229	150	122	115
	No. Reject	4596	1613	1053	520	400	395
	LB vs LB CPLEX 5h (%)	3.16	1.93	25.42	12.38	15.07	8.99
	UB vs UB CPLEX 5h (%)	−23.61	−30.41	−27.34	−23.71	−22.45	−21.95

Table 7: Performance of proposed branch-and-check and branch-and-solve approaches.

5.2. Contribution of proposed components

This section examines the contribution of the proposed components. All results were obtained by using the basic branch-and-check algorithm and additional components. Table 8 details the contributions of the cuts (38) and (39). The evaluation was done by using the basic branch-and-check algorithm since it gives a clear view of how the cuts work. Recall that TTDP-S(V_s) is actually a variant of the traveling salesman problem with time windows (TSPTW), which is an NP-hard problem even in the sense of finding feasible solutions (Savelsbergh (1985)). We obtain candidate solutions with checking steps that are very costly in terms of execution time when cuts (38) and (39) are not used. Applying these two cuts significantly accelerates the checks, resulting in more candidate solutions to be checked and thereby improving the performance of the proposed algorithm. This approach also raises the question of how to efficiently solve small-size traveling salesman problems with time windows (e.g., constraint programming could be a useful approach).

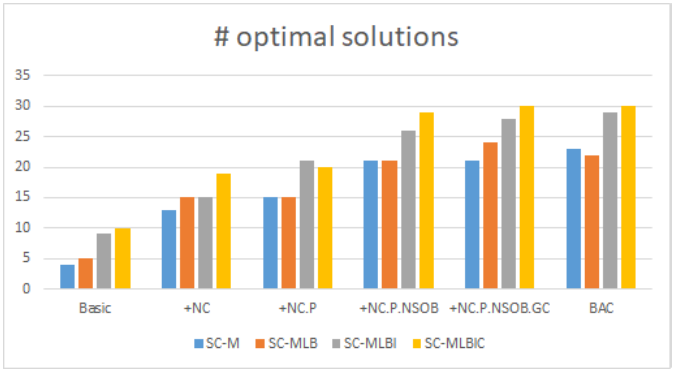
		SC-M	SC-MLB	SC-MLBI	SC-MLBIC
Without (38) & (39)	No. Fea. Sol.	5	5	8	11
	No. Check	25 403	25 446	20 247	19 044
With (38) & (39)	No. Fea. Sol	8	9	17	21
	No. Check	68 957	63 200	62 329	60 785
UB improvement (%)		−2.80	−5.40	−5.62	−5.89

Table 8: Contribution of valid inequalities (38) and (39) to strengthening the upper bound.

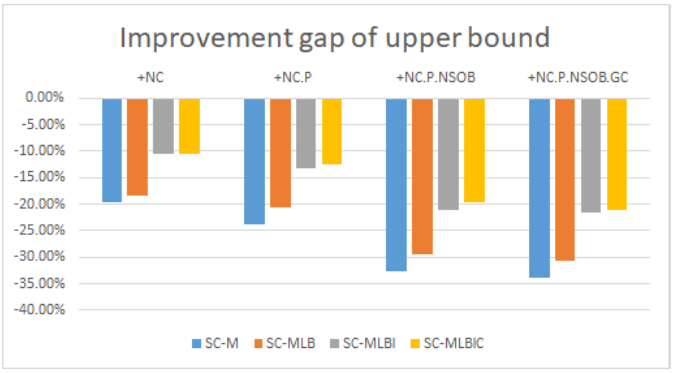
Next, Table 9 examines the contributions of each algorithmic component proposed. The basic BaC algorithm (Algorithm 1) is denoted "BASIC." The cut (33), preprocessing (Section 4.2.1), node selection and branching strategy (processing an internal node in Section 4.2), greedy checking (Algorithm 3), and the local branching and variable neighborhood search (Section 4.2.3) are denoted "NC," "P," "NSOB," "GC," and "LB," respectively. The configuration "+NC" thus means that we implement the basic branch-and-check plus the new cut component, the configuration "+NC.P" means that we implement the basic branch-and-check plus the new cut component and the preprocessing components, etc. The configuration "+NC.P.NSOB.GC.LB" is actually the proposed branch-and-check and is denoted "BaC."

Table 9A describes how each algorithmic component contributes to the number of solved instances. The four left-most columns present the number of optimal solutions in which the basic BaC is proven for each set. Each of the next four columns presents the number of optimal solutions returned by a configuration consisting of the BaC and new algorithmic components. Table 9B compares the upper bound obtained by using the new algorithmic components with that obtained by using the basic approach. The numbers show the important contribution of each algorithmic idea in terms of improving the upper bounds, especially valid inequalities, preprocessing, and the branching- and node-selection strategy. Table 9C shows that the feasibility issue can be solved

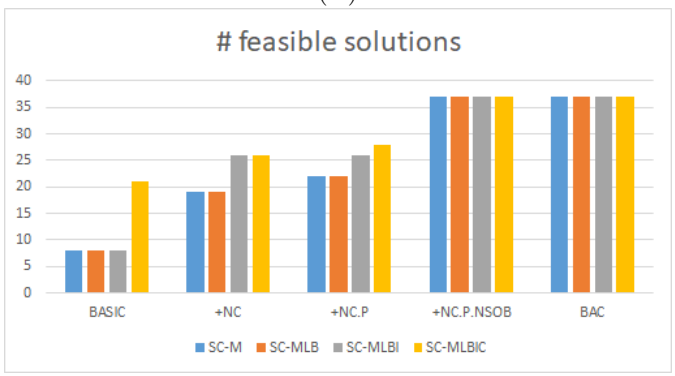
either by a local branching and variable neighborhood search or by a node selection and branching strategy (the configuration + NC.P.NSOB—where we also check the feasibility for partial solutions; see line 17). However, note that the local branching and variable neighborhood search procedure generates solutions with, on average, 2.90% improvement for lower bounds compared with solutions generated by the +NC.P.NSOB configuration, as indicated in Table 10. Also, this approach helps reduce overall execution time because finding good solutions earlier helps exclude poor candidates from consideration. Finally, Table 9D reports the average execution time in seconds for both solved and unsolved instances. The results show that the average running time is reduced when a new component is added and that all algorithmic components contribute to the improved performance of the basic branch-and-check approach.



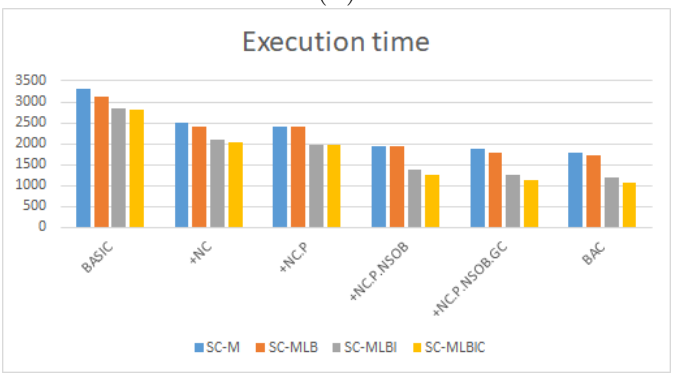
(A)



(B)



(C)



(D)

Table 9: Summary of contributions of each algorithmic component.

	SC-M	SC-MLB	SC-MLBI	SC-MLBIC
Improvement of LB (%)	3.02	2.42	3.19	2.97

Table 10: Average improvement of lower bounds upon using local branching and variable neighborhood component.

6. Conclusion

As part of the Smart Loire Project, this paper studies an orienteering problem called the tourist-trip design problem with rich constraints. The TTDP consists in delivering a route plan to tourists according to their preferences, budget, time constraints, and tourism constraints. Unlike most studies in this field, we consider not only the most common characteristics such as duration, budget, and category constraint but also practical constraints, including implication rules, precedence rules, and exclusion rules.

We propose an exact method based on the branch-and-check approach to solve the problem. The experiment with 37 new instances shows that the proposed approach with customized algorithmic components thoroughly outperforms CPLEX.

A possible direction for future research is to extend the method to solve the multiple-period tourist trip design problem where, instead of generating a single tour, multiple tours are generated for several days. We also plan to develop an efficient meta-heuristic approach to rapidly provide good feasible solutions for tourists. The algorithm proposed in this paper can then be used to evaluate the performance of this heuristic approach.

Acknowledgments

This research was partially funded by the Région Centre - Val de Loire (France) through the project SmartLoire.

References

Cordeau, J., Gendreau, M., Laporte, G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30 (2), 105–119.

Dumas, Y., Desrosiers, J., Gélinas, É., Solomon, M. M., 1995. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 43 (2), 367–371.

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming 98 (1), 23–47.

Froger, A., Gendreau, M., Mendoza, J. E., Pinson, E., Rousseau, L.-M., 2017. A branch-and-check approach for a wind turbine maintenance scheduling problem. Computers & Operations Research 88, 117–136.

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.

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

Lam, E., Van Hentenryck, P., 2017. Branch-and-check with explanations for the vehicle routing problem with time windows. In: International Conference on Principles and Practice of Constraint Programming. CP 2017. Lecture Notes in Computer Science, vol 10416. Springer, Cham. Springer, pp. 579–595.

Lin, S., Yu, V. F., 2017. Solving the team orienteering problem with time windows and mandatory visits by multi-start simulated annealing. Computers & Industrial Engineering 114, 195–205.

Lin, S.-W., Vincent, F. Y., 2015. A simulated annealing heuristic for the multiconstraint team orienteering problem with multiple time windows. Applied Soft Computing 37, 632–642.

Lu, Y., Benlic, U., Wu, Q., 2018. A memetic algorithm for the orienteering problem with mandatory visits and exclusionary constraints. European Journal of Operational Research 268 (1), 54–69.

Mladenovic, N., Hansen, P., 1997. Variable neighborhood search. Computers & Operation Research 24 (11), 1097–1100.

Palomo-Martínez, P. J., Salazar-Aguilar, M. A., Laporte, G., Langevin, A., 2017. A hybrid variable neighborhood search for the orienteering problem with mandatory visits and exclusionary constraints. Computers & Operations Research 78, 408–419.

Sadykov, R., 2008. A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates. European Journal of Operational Research 189 (3), 1284–1304.

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

Solomon, M. M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research 35 (2), 254–265.

Souffriau, W., Vansteenwegen, P., 2010. Tourist trip planning functionalities: State–of–the–art and future. In: Daniel, F., Facca, F. M. (Eds.), Current Trends in Web Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 474–485.

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.

Thorsteinsson, E. S., 2001. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: International Conference on Principles and Practice of Constraint Programming. CP 2001. Lecture Notes in Computer Science, vol 2239. Springer, pp. 16–30.

Tricoire, F., Romauch, M., Doerner, K. F., Hartl, R. F., 2010. Heuristics for the multi-period orienteering problem with multiple time windows. Computers & Operations Research 37 (2), 351–367.

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

Vu, D. M., Hewitt, M., Boland, N., Savelsbergh, M., 2019. Dynamic discretization discovery for solving the time dependent traveling salesman problem with time windows. Transportation Science. In press.