Solving the Traveling Salesman Problem with release dates via branch and cut

Agustín Montero[a,*], Isabel Méndez-Díaz[a,b], Juan José Miranda-Bront[c,d,*]

[a]*Departamento de Computación, FCEN, Universidad de Buenos Aires, Argentina*
[b]*Instituto de Investigación en Ciencias de la Computación (ICC), CONICET-UBA, Argentina*
[c]*Universidad Torcuato Di Tella, Buenos Aires, Argentina*
[d]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina*

Abstract

In this paper we study the Traveling Salesman Problem with release dates (TSP-rd) and completion time minimization. The TSP-rd considers a single vehicle and a set of customers that must be served with goods that arrive to the depot over time, during the planning horizon. The time at which each requested good arrives is called *release date* and it is known in advance. The vehicle cannot depart to serve a customer before the associated release date. Thus, the release date of the customers in each route must not be larger than the starting time of the route. The objective is to determine a set of routes for the vehicle, starting and ending at the depot, where the completion time needed to serve all customers is minimized. We propose a new Integer Linear Programming model and develop a branch and cut algorithm with tailored enhancements to improve its performance. The algorithm proved to be able to significantly reduce the computation times of the best exact algorithm in the literature, obtaining 24 new optimal solutions on benchmark instances with up to 30 customers within one hour. We further extend the benchmark to instances with up to 50 customers where the algorithm proved to be efficient. Building upon these results, the proposed model is adapted to new TSP-rd variants (Capacitated and Prize-Collecting TSP), with different objectives: completion time minimization and traveling distance minimization. To the best of our knowledge, our work is the first in-depth study to report extensive results for the TSP-rd through a branch and cut, establishing a baseline and providing insights for future approaches. Overall, the approach proved to be very effective and gives a flexible framework for several variants, opening the discussion about formulations, algorithms and new benchmark instances.

Keywords: Traveling Salesman Problem, Release Dates, Integer Linear Programming, Branch and Cut

*Corresponding author
Email addresses: `aimontero@dc.uba.ar` (Agustín Montero), `imendez@dc.uba.ar` (Isabel Méndez-Díaz), `jmiranda@utdt.edu` (Juan José Miranda-Bront)

1. Introduction and literature review

In this paper, we address the Traveling Salesman Problem with release dates and completion time minimization (TSP-rd(time)) with an exact approach. The TSP-rd(time) tackles a key operational constraint within nowadays last-mile logistics, partly motivated by same-day and fast deliveries. The package requested by a customer is not necessarily available at the beginning of the planning horizon, representing the timing of its arrival to the distribution center. Thus, the vehicle is allowed to perform multiple routes in order to serve all the customers. The TSP-rd(time) is formulated as a synchronization problem, and so far neglects the effect of the vehicle capacity. Variants of routing problems with release dates have only recently been introduced in the literature and applications arise in the context of cross docking and same-day delivery problems (Mor et al. [13]).

The motivation behind release dates is to represent the time at which a requested package arrives at the depot. In this fashion, it forces the vehicle to only satisfy the requests for customers whose packages are ready at the depot at the moment of departure. This version of release dates is first introduced by Cattaruzza et al. [10] where the authors tackle the multi-vehicle routing problem with time windows and release dates. The objective is to minimize the total travel distance, and the authors propose a hybrid genetic algorithm on a set of instances adapted from Solomon [17]. A single-vehicle variant of a routing problem with release dates is presented in Archetti et al. [5]. The problem is called Traveling Salesman Problem with release dates (TSP-rd) for the first time by the authors and it does not consider capacities or time windows. Two different objective functions are proposed: (a) completion time minimization (TSP-rd(time)), where the idea is to minimize the time needed to complete the distribution of all packages computed as the sum of the total traveling time plus the total waiting time, and (b) total traveling distance minimization (TSP-rd(distance)), in which there is a deadline for completing the distribution and the goal is to minimize the total traveling distance. Although both variants are NP-hard, the authors show that they can be solved in polynomial time provided that the underlying graph has a special structure. Reyes et al. [15] generalize these results considering a service guarantee that implies a common deadline for the orders after each release date. Still the contributions are towards the complexity of the problem on specific networks (e.g., the half-line). Archetti et al. [4] present an Integer Linear Programming (ILP) formulation for the TSP-rd(time) and propose two iterated local-search procedures based on a destroy-and-repair scheme. The article focuses on the heuristics and report results for instances up to 500 customers. A natural extension for multi-vehicle routing problem with release dates is presented in Shelbourne et al. [16] for which a path relinking algorithm is proposed. Due dates are considered for each order, i.e. a time by which the order must be delivered to the customer. The objective function considered combines an operational cost and customer service level by means of the total distance traveled and the total weighted tardiness of delivery, respectively. Waiting times are not considered and no results are reported for the single-vehicle case. Another version which is related to TSP-rd is the Multi-Trip Vehicle Routing Problem (MTVRP) where each vehicle is allowed to perform multiple trips starting and ending at the depot due to duration or capacity constraints (see Azi et al. [6, 7, 8] for potential applications and results that, despite missing release dates, incorporate time windows). An exact solution framework which accounts for the modeling of release dates for the Capacitated MTVRP with Time Windows (CMTVRP-TW) is proposed in Paradiso et al. [14]. It relies on column generation, column enumeration and cutting planes. Among the four different variants, one of them incorporates release dates. However, the objective function accounts for the minimization of the total traveled distance instead of the makespan and the developed labeling algorithm explicitly exploits the presence of time windows. Thus, no direct comparison with Archetti et al. [4] is established, as the framework cannot be directly adapted.

Finally, a survey about routing problems over time is presented by Mor et al. [13], and a recent article about challenges in routing and inventory routing in the context of e-commerce and last-mile delivery is presented in Archetti et al. [3], including a dedicated section about release dates.

Release dates are still relatively new in the VRP literature. The contributions of our paper are threefold. First, on the methodological side, we propose a new ILP formulation for the TSP-rd(time) where the multiple trips are modelled via an adaptation of the Generalized Cut Set (GCS) constraints (see, e.g., Taccari [18]). To the best of our knowledge, the only exact approach for the TSP-rd(time) is proposed in Archetti et al. [4]. Although both models consider the edge flow variables, our approach provides an improved fashion to model the multiple visits to the depot. In addition, we propose two enhanced families of valid inequalities to model the interaction among the release dates. Second, from an algorithmic standpoint, we develop a tailored branch and cut algorithm incorporating the new valid inequalities as part of the formulation, the GCS, an initial heuristic to compute an upper bound on the instance and a specific branching criterion. We conduct extensive computational experiments over benchmark instances and compare our results with the ones reported by Archetti et al. [4]. We show that our method outperforms their approach, and we provide strong evidence on the components that drive such improvement via specific experiments. We provide 24 new optimal solutions for the benchmark instances having up to 30 customers proposed in Archetti et al. [4], and we further expand the benchmark and report results for instances with up to 50 customers. Third, building upon the previous results, we consider different variants for the TSP-rd by incorporating other characteristics such as capacities, distances and profits. The model is adapted to each variant and we generate tailored instances in each case to study the performance of our algorithm in different setups, which may result very valuable for practitioners or researchers tackling such problems. Overall, our paper contributes with improved methodology and strong computational results for a family of single vehicle routing problems with release dates.

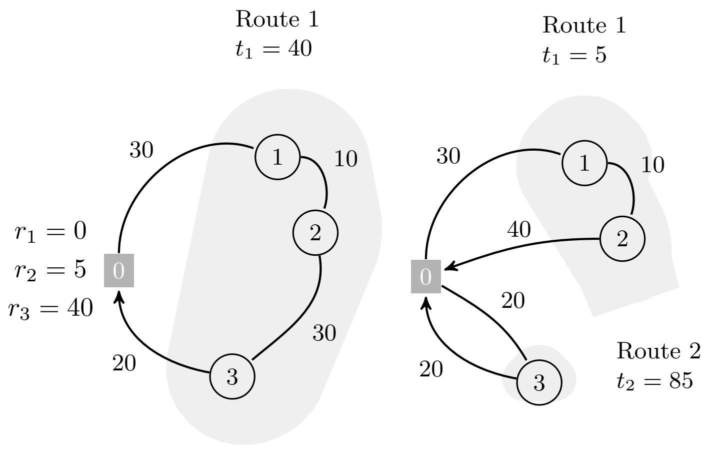
The rest of the paper is organized as follows. In Section 2 we present the formal definition of the TSP-rd(time) and the notation used along the paper. In Section 3 we introduce the formulation proposed by Archetti et al. [4] and our new formulation. Section 4 describes the details of the BC algorithm based on the new formulation, and Section 5 reports the computational results for the TSP-rd(time). Several TSP-rd variants are explored in Section 6, considering both completion time and distance minimization, as well as capacities and the prize-collecting version of the problem. Finally, we conclude and state some future research lines in Section 7.

2. Problem definition

Let $G = (V, A)$ be a complete digraph, with V the set of vertices and A the set of edges. The set $V = \{0\} \cup N$ models the depot, denoted by vertex 0, and the set of customers $N = \{1, \ldots, n\}$. We consider a traveling time t_{ij} associated to each edge $(i, j) \in A$ which satisfy the triangle inequality. We assume a non-negative release date r_i for each customer $i \in N$, which represents the time at which the requested package arrives at the depot. In particular, setting $r_i = 0$ models that the package is available at the beginning of the distribution because it arrived overnight. Note that the classical TSP can be retrieved by setting $r_i = 0$ for all $i \in N$. The operations are carried out by one vehicle with infinite capacity, ready to depart at $t = 0$. The vehicle is allowed to perform multiple consecutive routes. However, each route can only include packages which are ready before the corresponding departure from the depot (i.e., its release date r_i is at most the departure time of the vehicle from depot). The objective is to serve all customers at minimum total completion time, defined as the time at which the vehicle is back to the depot after visiting all customers, computed as the sum of travel and waiting times.

For simplicity, we adopt the definition of a route used in Archetti et al. [4]. A *route* refers to a trip that starts and ends at the depot and that does not visit the depot in between. Note that in the context of multiple vehicles (e.g., Cattaruzza et al. [10]) trips and routes may refer to different concepts.

Figure 1 shows two examples of feasible solutions for a distribution network with 3 customers. Let t_k be the starting time of the k-th route within a solution. In the example, Solution (1a) involves one route which departs as soon as all packages are available at time $40 = \max\{r_1,\, r_2,\, r_3\}$, resulting in a completion time of 130 and a traveled time of 90. Solution (1b) involves two routes. In the first one, the vehicle departs at time $5 = \max\{r_1,\, r_2\}$ being able to deliver the requested package of Customers 1 and 2. As soon as the vehicle returns to the depot at time 85, the requested package of Customer 3 is available ($r_3 < 85$), and the vehicle departs again to visit the remaining customer with the second route. The total completion time is 125 with a traveled time of 120 and a waiting time of 5. In this example, Solution (1b) is better than Solution (1a) as the total completion time of the former is lower.



(a) Solution A (b) Solution B

Figure 1: Examples of feasible solutions for the TSP-rd(time).

We introduce two properties presented by Archetti et al. [4], which are important to understand the structure of the TSP-rd(time) and to enhance the mathematical formulations presented in Section 3.

Property 1 (No waiting time after first departure)**.** *Given an instance of the TSP-rd(time), there exists an optimal solution with no waiting time after the departure of the first route.*

The intuition behind this result is that each departure can be shifted forward without changing the set of customers served in each route, neither the total completion time as depicted in Figure 2.

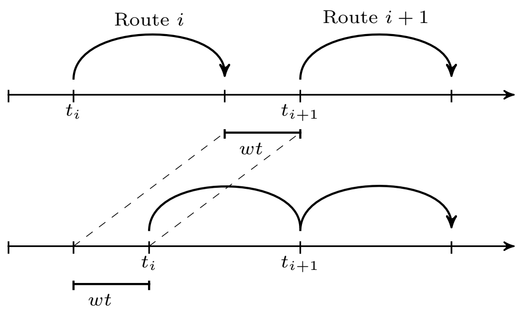


Figure 2: Forward shift of Route i to remove waiting time after departure (Property 1)

Property 2 (Routes & latest release date)**.** *There exists an optimal solution with exactly one route starting not earlier than r_{max}, i.e., the latest release date.*

We omit the proof and refer the reader to Archetti et al. [4] for further details.

3. ILP formulations

In this section we present two ILP formulations for the TSP-rd(time). First, we describe the formulation proposed in Archetti et al. [4]. We then introduce our new formulation as well as an improved version with tightened constraints.

3.1. AFMG formulation (Archetti et al. [4])

The 3-index formulation introduced in Archetti et al. [4] for the TSP-rd(time), named *AFMG*, is based on flow variables indexed by the route in which the edge is traversed. Let K be a set of routes, with $|K|$ being an upper bound for the number of routes in the optimal solution. Define binary variables x_{ij}^k taking value 1 if and only if edge $(i,j) \in A$ is traversed in route $k \in K$, and let binary variables y_k^i take value 1 if and only if vertex $i \in V$ is visited in route $k \in K$. To account for the timing of the route, let t_{start}^k and t_{end}^k denote the starting and ending time of route $k \in K$, respectively.

Additional continuous non-negative flow variables u_{ij}^k are added to enforce subtour elimination, and a binary variable x_{00}^k is considered for each $k \in K$, taking the value 1 if and only if route k is an empty route, i.e., it visits no customers. As a result, the model is:

$$\min\ t_{end}^{|K|} \tag{1}$$

s.t.

$$\sum_{k\in K} y_i^k = 1 \quad \forall\ i \in N \tag{2}$$

$$\sum_{j\in V} x_{ij}^k = \sum_{j\in V} x_{ji}^k = y_i^k \quad \forall\ i \in V, \forall\ k \in K \tag{3}$$

$$\sum_{j\in V} u_{ji}^k - \sum_{j\in V} u_{ij}^k = y_i^k \quad \forall\ i \in N, \forall\ k \in K \tag{4}$$

$$u_{ij}^k \leq (n-1)\, x_{ij}^k \quad \forall\ (i,j) \in A, \forall\ k \in K \tag{5}$$

$$t_{end}^k = t_{start}^k + \sum_{(i,j)\in A} t_{ij}\, x_{ij}^k \quad \forall\ k \in K \tag{6}$$

$$t_{end}^k \leq t_{start}^{k+1} \quad \forall\ k \in K \setminus \{|K|\} \tag{7}$$

$$t_{start}^k \geq r_i\, y_i^k \quad \forall\ k \in K, \forall\ i \in N \tag{8}$$

$$t_{end}^k = t_{start}^{k+1} \quad \forall\ k \in K \setminus \{|K|\} \tag{9}$$

$$t_{start}^k \leq r_{\max} \quad \forall\ k \in K \setminus \{|K|\} \tag{10}$$

$$x_{ij}^k \leq 1 - x_{00}^k \quad \forall\ (i,j) \in A, \forall\ k \in K \setminus \{|K|\} \tag{11}$$

$$x_{00}^k \geq x_{00}^{k+1} \quad \forall\ k \in K \setminus \{|K|\} \tag{12}$$

$$x_{ij}^k \in \{0,1\} \quad \forall\ (i,j) \in A, \forall\ k \in K \tag{13}$$

$$y_i^k \in \{0,1\} \quad \forall\ i \in V, \forall\ k \in K \tag{14}$$

$$t_{start}^k,\ t_{end}^k \geq 0 \quad \forall\ k \in K \tag{15}$$

$$u_{ij}^k \geq 0 \quad \forall\ (i,j) \in A, \forall\ k \in K \tag{16}$$

The objective function (1) minimizes the total completion time as it is the ending time of the last route. Constraints (2) guarantee that all customers are visited by exactly one route. Constraints (3) establish the flow conservation between edges entering and leaving each customer, and connect flow variables with indicator variables y_i^k. Constraints (4)-(5) impose that each route is a circuit connected to the depot. In particular, constraints (5) were first proposed by Gavish and Graves [11] and are used to prevent subtours through a flow that decreases while the vehicle visits customers. The relation between variables t_{start}^k and t_{end}^k is set by constraints (6) and (7). Feasible starting times of a route depending on the release dates of the customers served is modeled through constraints (8). Properties 1 and 2 are incorporated by constraints (9) and (10) respectively in order to reinforce the formulation. Note that constraints (9) imply (7), but we include both sets of constraints to be consistent with the model presented by Archetti et al. [4]. Constraints (11) enable an edge by a tour only if the latter visits at least one vertex. Finally, constraints (12) remove symmetric solutions by setting that if route k is empty then all routes $\tilde{k} < k$ must be empty as well. Note that all empty routes, if any, will precede all the non-empty routes given constraints (12), and the fact that the last route (noted as $|K|$) departs not earlier than $r_{\max}$ due to constraints (10).

Although the model uses an upper bound on the number of routes in the optimal solution, Archetti et al. [4] develop an exact algorithm by setting $|K| = n$. The authors only replace it with a tighter value for the purpose of embedding the model in a heuristic scheme.

3.2. AJI formulation

We propose a new formulation that also uses the 3-index flow variables indexed by route in which the edge is traversed. However, similar to other TSP variants, modelling specific constraints in a different fashion can translate into improved formulations that perform better in practice. This is the aim of our formulation, where subtours are prevented by an adaptation of the GCS constraints. Thus, we do not consider variables u_{ij}^k as part of the formulation. Moreover, both t_{start}^k and t_{end}^k are removed and we let continuous variables t_k indicate the starting time of route $k \in K$, and consider the special case of $t_{|K|+1}$ that indicates the ending time of route $|K|$. For the sake of notation, given $S \subseteq V$, let $\delta^+(S) = \{(i,j) \in A\ :\ i \in S, j \in V \setminus S\}$. Then, the formulation reads:

$$\min\ t_{|K|+1} \tag{17}$$

$$\sum_{k\in K} y_i^k = 1 \quad \forall\, i \in N \tag{18}$$

$$\sum_{(i,j)\in A} x_{ij}^k = \sum_{(i,j)\in A} x_{ji}^k = y_i^k \quad \forall\, i \in V, \forall\, k \in K \tag{19}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij}^k \geq y_l^k \quad \forall\, l \in S \subseteq N, |S| \geq 2, \forall\, k \in K \tag{20}$$

$$y_0^k \leq y_0^{k+1} \quad \forall\, k \in K \setminus \{|K|\} \tag{21}$$

$$t_{k+1} = t_k + \sum_{(i,j)\in A} t_{ij}\, x_{ij}^k \quad \forall\, k \in K \tag{22}$$

$$t_k \geq r_i\, y_i^k \quad \forall\, k \in K, \forall\, i \in N \tag{23}$$

$$t_k \leq r_{\max} \quad \forall\, k \in K \setminus \{|K|\} \tag{24}$$

$$x_{ij}^k \in \{0,1\} \quad \forall\, (i,j) \in A, \forall\, k \in K \tag{25}$$

$$y_i^k \in \{0,1\} \quad \forall\, i \in V, \forall\, k \in K \tag{26}$$

$$t_k \geq 0 \quad \forall\, k \in K \tag{27}$$

$$t_{|K|+1} \geq 0 \tag{28}$$

The objective function (17) minimizes the total completion time as it is the ending time of the last route, i.e., route $|K|$. Constraints (18)-(19) play the same role as in the formulation from Archetti et al. [4] to ensure that all customers are visited and that flow conservation is satisfied. Constraints (20) are based on Generalized CutSet Inequalities (GCS) to enforce elimination of subtours. Symmetry breaking is done through constraints (21) which establish that all empty routes, if any, precede all the non-empty routes. Note that variables x_{00}^k are not needed because of variables y_0^k and constraints (21). Constraints (22) are similar to constraints (6), but also incorporate the idea behind (9) and (7) that exploit Property 1. As a consequence, the model removes solutions with waiting time after the first departure. Although it is not needed, such solutions with waiting time can be included by relaxing constraints (22) as inequalities. Finally, constraints (23) and (24) are the analogous of constraints (8) and (10).

Regarding the set of routes K, the model is still flexible as it makes use of an upper bound on the number of routes in the optimal solution. However, in all cases, we set $|K| = n$ which is the trivial upper bound on the number of routes.

3.3. Valid inequalities

Let $K_{\leq k} = \{h \in K \;:\; h \leq k\}$ denotes the subset of route indices smaller than or equal to k. We provide in this section two families of valid inequalities to strengthen the AJI formulation. Intuitively, the first family states that once a customer $i \in N$ is visited, all subsequent routes must start later than release date r_i.

Proposition 1. *Given a customer $i \in N$ and a route $k \in K$, constraint*

$$t_k \geq r_i \sum_{h \in K_{\leq k}} y_i^h \tag{29}$$

is valid for the AJI formulation.

Proof. Constraints (23) can be generalized by considering $h \leq k$, i.e. $t_k \geq r_i y_i^h$. Since at most one y_i^h can take value 1 for $h \in K_{\leq k}$, (23) can be lifted by adding the rhs of these constraints, resulting in the desired inequality. $\square$

The intuition behind the second family is to exploit that empty routes, if any, precede non-empty routes. Then, route $k \in K$ is used (i.e., $y_0^k = 1$), if at least one customer is assigned to k or to a precedent route $h < k$.

Proposition 2. *Given a customer $i \in N$ and a route $k \in K$, constraint*

$$\sum_{h \in K_{\leq k}} y_i^h \leq y_0^k \tag{30}$$

is valid for the AJI formulation.

Proof. We separate the proof in two cases. If the lhs is 0, the inequality is trivially satisfied. Otherwise, note that at most one of the variables takes value 1 due to constraints (18). Let $h' \leq k$ be the index of the trip visiting i. Then, $y_i^{h'} = y_0^{h'} = 1$ due to constraints (20) and (19). If $h' < k$, then by constraints (21) we get $y_0^k = 1$ as well, which concludes the proof. $\square$

Constraints (23) are replaced by its strengthened version (29). Constraints (30) are incorporated as part of the formulation, although they are not needed for the formulation to be correct. Both (29) and (30) should help by removing fractional points and improving the linear relaxation. We call the resulting model *AJI++*.

4. Branch and Cut algorithm

We develop a branch and cut (BC) algorithm based on the formulation presented in Section 3.2 for the TSP-rd(time). In this section we describe some of the main components, such as how we compute an initial feasible solution, the cutting plane algorithm and the branching scheme.

4.1. Initial feasible solution

The BC algorithm is initialized with a feasible solution obtained by a *time-explorer* multi-start heuristic depicted in Algorithm 1. The heuristic considers as input a set T of feasible departure-times for the first route of the TSP-rd(time) solution, and the idea is to compute a sequence of tours using the *myopic*-optimal solution for each $t \in T$. In other words, every time the vehicle reaches the depot, it either departs to serve the customers whose goods arrived while the vehicle was traveling, or waits for the next customer that can be served and then immediately departs. The routing (i.e., the order in which customers are going to be served), is decided by solving to optimality a TSP instance defined by the underlying sub-graph.

Algorithm 1 Time-Explorer heuristic

Input: Instance $\mathcal{I}$ of the TSP-rd(time), with $G = (V, A)$, $V = N \cup \{0\}$, travel times t_{ij} for $(i, j) \in A$
Output: Feasible solution x_{best} of cost z_{best}

```
1. Initialization. Define $T = \{r_{\min}, \dots, r_{\max}\}$ as the list of possible departure times
   for the first trip of the depot as all the (integer) instants between the minimum
   and maximum release dates, where $r_{\min} = \min_{i \in N} r_i$ and $r_{\max} = \max_{i \in N} r_i$.
   Initialize the cache of TSP solutions $mem = \{\}$, indexed by subsets of vertices;
   current solution information $z_{best} = \infty$ and $x_{best} = nil$.
2. Iteration. If $T$ is non-empty, get the next $t \in T$ and define the current solution
   $x =< 0 >$ with makespan $z = t$. Set $NS = N$ as the set of non-assigned customers.
3. Auxiliary TSP. Compute $S = \{v \in NS \mid r_v \leq z\}$ as the set of non-visited customers
   ready to be delivered at time $z$. Let $\mathcal{I}_S$ be the auxiliary TSP instance defined by the
   subset of vertices $S \cup \{0\}$. If $mem[S]$ is defined, retrieve the optimal solution $x_{tsp}$
   and its objective value $z_{tsp}$. Otherwise, solve the auxiliary TSP instance, obtain
   $x_{tsp}$ and $z_{tsp}$, and update $mem[S] = (x_{tsp}, z_{tsp})$ for future iterations.
   Update the current solution $x = x + x_{tsp}$ and $z = z + z_{tsp}$ as the current makespan.
4. Feasibility check. Update $NS = NS \setminus S$. If $NS$ is non-empty, update $z =$
   $\max\{z, \min_{v \in NS}\{r_v\}\}$, and return to Step 3.
5. Termination. If $z < z_{best}$, then update $z_{best} = z$ and $x_{best} = x$. Set $T = T \setminus \{t\}$. If
   $T$ is non-empty, go to Step 2. Otherwise, return the best solution found $x_{best}, z_{best}$.
```

Algorithm 1 starts with the set T of all the (integer) time instants between the minimum and maximum release dates, as there is no need to consider values outside that range. During the execution, it is possible for a TSP instance to appear as an auxiliary subproblem more than once. In order to avoid solving the same subproblem several times and speed-up the execution, the algorithm stores a table that maps TSP instances (i.e., subsets of vertices) to optimal solutions. In Step 3, every time a TSP is formulated, this table is first examined to check whether the same instance has been solved previously and, in that case, retrieves the optimal solution. As the number of vertices n is at most 50, this table can be queried and modified efficiently.

We remark that it is not sufficient to consider in T only the release dates of the vertices, since eventually a better solution can be obtained by initially departing from the depot at other time. For example, consider an instance with 3 customers where $t_{ij} = 2$ for all $(i, j) \in A$. Let the release dates be 0, 4 and 5, respectively. Then, the best solution that the Time-Explorer heuristic can provide is obtained by departing at $t = 1$, which is not a release date. However, during the execution it is possible to identify time instants that will not lead to an improving solution and, therefore, can be removed from T as starting values in Step 5. One of those situations involves the first auxiliary TSP considered and eventual waiting times. Let t be the current initial departure time from the depot and τ the return to the depot after the first tour. If the latter occurs before the minimum release date of the non-visited vertices, i.e. $\tau < \min_{i \in NS} r_i$, the solution would include waiting times. Then, the heuristic can remove the interval $[t, t + \min_{i \in NS} r_i - \tau]$ from T since the subset of vertices in the underlying first TSP instance remains unchanged for those time instants. A second situation arises when the initial departure time t occurs before the maximum release date, but the return to the depot τ afterwards, i.e., $t \leq r_{\max} \leq \tau$. In these cases, the heuristic does not need to consider initial departures $t' > t$ since the subset of vertices in the underlying first TSP instance remains unchanged for those time instants. Therefore, the subsequent TSP instances, if any, also remain unchanged and only lead to solutions at most as good as the one obtained from the initial depart at t in the best case.

In practice, Concorde [2] is used to solve the auxiliary TSP instances, and a feasible solution is guaranteed (e.g., departing initially at $t = r_{\max}$). The speed-up is driven mostly by exploiting the cache of TSP solutions, and the heuristic runs in less than one second per instance. These running times to compute an initial upper bound are negligible in the context of the BC algorithm developed, specially for the most demanding instances.

4.2. Cutting planes

The family of GCS inequalities (20) is exponential, and therefore we incorporate them as cuts through the corresponding separation algorithm. In this case, violated cuts can be found by solving a sequence of polynomial-time max-flow sub-problems (see, e.g. Taccari [18]). Constraints (20) are needed in the formulation to be correct, so we need to incorporate them as lazy-cuts, i.e., on every integer solution. Moreover, such constraints can be used to tighten the model by removing fractional points. In particular, we found that this type of cuts are more effective when added at the root node, while they downgrade the performance when added during node enumeration given that the trade-off between time required to separate them and improvement in the linear relaxation is not convenient. As a consequence, the upper limit on the number of cutting plane passes is set to 100 in the root node, and to 20 (i.e., CPLEX's default value) on the other nodes.

4.3. Branching scheme

During tree enumeration, decisions are made about which variable to choose to branch on at each node. A custom branching is considered by assigning priorities to variables (25) and (26). Variables y_0^k have highest priority as they determine whether a new route is used. Next in priority order, we have variables y_i^k for $i > 0$, as they decide if a customer is served or not in the route k. Finally, flow variables x_{ij}^k have the lowest priority. Within each group, all variables have the same priority and we let CPLEX apply its default criterion for the variable selection.

5. Computational results

5.1. Experimental setup

The proposed BC algorithm is evaluated on the instances considered in Archetti et al. [4] which are derived from Solomon [17]. These are adapted to the TSP-rd(time) by discarding time windows and considering different sets of clustered located ($C1$, $C2$), randomly located ($R1$) and a mix of randomly located and clustered located ($RC1$) customers. Sets $R2$ and $RC2$ were discarded because they have the same coordinates as $R1$ and $RC1$, respectively, and they only differ in the time windows information.

The instances are characterized by 3 values:

- n: number of customers
- d_{TSP}: optimal TSP value of the underlying graph
- β: parameter that controls the width of the interval in which release dates are defined

For each instance, the original data is truncated after $n + 1$ nodes, and the first node is set to be the depot. The release date of the i-th customer is determined by uniformly sampling an integer from $[0, \beta \times d_{TSP}]$. Instances are generated with $\beta = \{0.5, 1, 1.5, 2, 2.5, 3\}$ resulting in 24 instances for each $n = \{10, 15, 20, 25, 30, 50\}$.

In order to extend our results, we use the same approach to generate instances for $n = \{35, 40, 45\}$, resulting in 72 new instances. All instances are publicly available at `github.com/agusmontero/tsprd`.

The algorithms are implemented in the `C++` programming language using g++ 7.5.0 and an Ubuntu 18.04 LTS as operating system. The experiments are run on a workstation with an Intel Core i7-8700 3.20 GHz processor with 32 GB of RAM. `CPLEX 12.9` is used as optimization solver. We set the traditional branch and cut search strategy[1] and impose a total time limit of 3600 seconds for the execution time of each instance. The source code is avaiable at `github.com/agusmontero/tsprd`[2].

5.2. Comparison of formulations

We begin by comparing the performance of the AJI`++` model with the approach proposed in Archetti et al. [4] which is, to the best of our knowledge, the best exact method for the TSP-rd(time) in the scientific literature. For the sake of simplicity, in the remaining of the paper we refer to the methods presented through the article both as *formulations* and as *algorithms*. The former references the corresponding ILP model and the later the BC algorithm based on such ILP formulation. The following methods are considered:

- AFMG: ILP model proposed in Archetti et al. [4];

[1]https://www.ibm.com/docs/en/icos/12.9.0?topic=enumerations-ilocplexmipsearch
[2]Upon acceptance of the article.

- AJI++: ILP formulation from Section 3.2 incorporating the GCS for the subtour elimination, without initial heuristic and CPLEX default branching strategy.

To account for the different machines used[3], time units are scaled in AJI++. Table 1 reports the number of instances solved to proven optimality within 3600 seconds considering the time scaling. Our model is able to solve all instances up to $n = 20$, 67% of the instances for $n = 25$ and 42% of the instances for $n = 30$ within the time limit. Moreover, AJI++ is also able to solve a subset of instances up to $n = 50$. It improves the best known results[4] of Archetti et al. [4], for which authors report to solve 96%, 54% and 29%, respectively, highlighting the effectiveness of our proposal.

Building upon this initial comparison regarding the effectiveness, we focus on the AJI++ to assess regarding the impact of the different components affecting the BC algorithm.

Source	n	AFMG		AJI++	
		Solved	% Solved	Solved	% Solved
	10	24	100%	24	100%
	15	24	100%	24	100%
Archetti et al. [4]	20	23	96%	24	100%
	25	13	54%	16	67%
	30	7	29%	10	42%
	35	*n.r.*	*n.r.*	6	25%
New instances	40	*n.r.*	*n.r.*	3	13%
	45	*n.r.*	*n.r.*	3	13%
Archetti et al. [4]	50	*n.r.*	*n.r.*	3	13%

Table 1: Number of TSP-rd(time) instances solved to proven optimality within 3600 seconds. Instances with $n \geq 35$ are not reported in Archetti et al. [4] which is indicated as *n.r.*

5.3. Impact of subtour elimination strategies

In this section, we focus on the impact of different alternatives to forbid subtours within each route in the solution. For this purpose, we consider the following variants of the AJI++ formulation:

- AJI: ILP formulation described in Section 3.2. We emphasize that constraints (29) and (30) are not part of this formulation.
- AJI poly-subtours: AJI formulation but replacing the GCS (20) by the polynomial family of constraints (4) - (5) used by Archetti et al. [4] for the AFMG model.
- AJI++ poly-subtours: Same modification as in the previous case, but for the AJI++ formulation. Observe that the difference between these two variants is the presence of constraints (29) and (30).

[3]Our processor is 43% faster than the one used in Archetti et al. [4] according to the CPU Mark index from `www.cpubenchmark.net` [1]

[4]Updated results were provided in a private communication.

Table 2 compares these three variants with the AJI++ formulation considered in Section 5.2. The key to the table is the following: the number of instances solved to proven optimality before the time limit (Solved), grouped by n; the average time (in seconds) required for the solved instances by the corresponding method, also grouped by n (Time); the average percentage gap of the best lower bound lb found by the corresponding method in the root node of the BC with respect to the best known solution (bks) for that instance, computed as $|bks - lb| \times \frac{100}{bks}$ (Root); the average percentage gap found during the complete execution of the BC algorithm (GAP).

The main message from Table 2 is that GCS-based subtour elimination constraints (20), when combined with constraints (29) and (30), result in the best method. More specifically, AJI++ solves 33 more instances to optimality within the imposed time limit than AJI++ poly-subtours in shorter computation times on average, and outperforming the other variants as well. In contrast, this behavior differs when considering the other two, more basic variants that do not consider the valid inequalities proposed in Section 3.3. In this case, AJI poly-subtours solves 4 more instances to optimality than AJI within the time limit. Based on limited additional experiments, we identified that the difference may be caused by a smaller number of general purpose cuts incorporated by CPLEX during the execution of the algorithm.

n	AJI poly-subtours				AJI				AJI++ poly-subtours				AJI++			
	Solved	Time	Root	GAP	Solved	Time	Root	GAP	Solved	Time	Root	GAP	Solved	Time	Root	GAP
10	24	2	6.82%	0.00%	24	2	8.14%	0.00%	24	1	2.76%	0.00%	24	0	0.49%	0.00%
15	24	309	15.41%	0.00%	22	349	14.97%	5.42%	24	38	12.06%	0.00%	24	3	4.82%	0.00%
20	9	995	20.18%	4.91%	7	145	24.38%	6.81%	13	365	16.58%	2.29%	24	128	11.81%	0.00%
25	3	1202	19.12%	10.35%	3	115	22.06%	10.55%	6	1010	15.65%	6.36%	18	788	13.52%	0.77%
30	1	2909	23.68%	16.72%	1	3451	26.46%	14.00%	3	812	18.61%	11.95%	13	1004	18.37%	3.65%

Table 2: Impact of adding GCS (20) subtour elimination constraints in combination with inequalities (29) and (30) when solving AJI++. Note that Column AJI++ corresponds to the same algorithm of Column AJI++ in Table 1. The difference that can be observed in Column Solved across the aforementioned tables is only due to the scaling needed in Table 1 to account for the different machine used in Archetti et al. [4]

This experiment also provides evidence on the impact of the valid inequalities proposed in Section 3.3. The results indicate that the incorporation of constraints (29) and (30) has a very positive effect, improving the root average percentage gap, solving more instances when comparing polynomial-size subtour elimination and GCS based formulation independently (9 for AJI poly-subtour vs. AJI++ poly-subtour; 46 for AJI vs. AJI++) in shorter average computation times and with smaller average final percentage gaps. Overall, AJI++ outperforms the other variants, appearing as a more robust method. Finally, we remark that this experiment is limited to instances having $n \leq 30$ as the other variants only solve a very limited number of instances to optimality for larger values of n.

5.4. Impact of the initial solution and the tailored branching strategy

Building upon the results in the previous section, we further investigate the impact of the other components developed for the BC algorithm. Using AJI++ as a baseline, we evaluate and quantify the benefit of incorporating the initial heuristic and the tailored branching strategy when considered in an isolated fashion as well as combined. For this experiment, we consider the following variants:

- AJI++ IS: AJI++ using an initial feasible solution obtained by Algorithm 1 and CPLEX default branching scheme.
- AJI++ Br: AJI++ using the tailored branching strategy described in Section 4.3. No initial solution considered.
- AJI++ IS+Br: A combination of the previous two variants, i.e. the AJI++ model using both the initial feasible solution obtained by Algorithm 1 and the tailored branching strategy from Section 4.3.

Table 3 summarizes the results obtained for these three variants as well as AJI++ over all the instances. The key to the table remains the same as in Table 2. Although it is not explicitly reported, Algorithm 1 is always able to obtain an initial feasible solution is less than 1 second with an average quality of 4.18% w.r.t. the best known solutions for $n \in \{10, \ldots, 50\}$. If considered independently, both AJI++ IS and AJI++ Br show improvements, the latter being more significant. For AJI++ IS, the addition of the warm-start mostly helps in reducing the GAP on average by 51% on the unsolved instances w.r.t. AJI++. AJI++ Br shows stronger improvements, reducing the GAP further on average by 58% but, most importantly, solving 28 additional instances with respect to AJI++. Finally, with both features activated, AJI++ IS+Br shows the best results by solving 30 additional instances and reducing the average GAP by 71% w.r.t. AJI++. The average computation time required by the BC is observed as increased (see, e.g., results for $n = 40$) due to the additional instances that are solved when features are incorporated. However, when analysing the results in AJI++ IS+Br only on instances solved by AJI++, the time required by the BC is reduced on average by 72%, showing that the former is faster on solving instances.

Finally, we remark that Archetti et al.[4] also generated instances for $n = 100$. Although the authors do not report results obtained with an exact algorithm for such instances, it is worth mentioning that Algorithm 1 is able to compute feasible initial solutions for all instances with $n = 100$. Briefly, it requires 13 seconds on average, having an average quality of 5.26% w.r.t. to the best known solutions reported in Archetti et al. [4], which are obtained via more sophisticated heuristic approaches. Regarding AJI++ IS+Br, we comment that it is not able to solve any instance for $n = 100$ within 3600 seconds.

Overall, the formulation AJI++ proves to be, in combination with the initial heuristic and the branching strategy, improving the current benchmark regarding exact algorithm in the literature studied by Archetti et al. [4]. Based on this

	AJI++			AJI++ + IS			AJI++ + Br			AJI++ + IS+Br		
n	Solved	Time	GAP	Solved	Time	GAP	Solved	Time	GAP	Solved	Time	GAP
10	24	0	0.00%	24	0	0.00%	24	0	0.00%	24	0	0.00%
15	24	3	0.00%	24	4	0.00%	24	2	0.00%	24	2	0.00%
20	24	128	0.00%	24	102	0.00%	24	20	0.00%	24	19	0.00%
25	18	788	0.77%	18	677	0.38%	23	255	0.12%	24	329	0.00%
30	13	1004	3.65%	11	488	2.77%	19	481	1.74%	19	579	1.13%
35	7	803	25.60%	7	705	11.99%	14	774	6.31%	14	1015	6.69%
40	3	276	32.09%	4	1105	14.73%	8	718	10.62%	9	776	9.11%
45	3	685	43.92%	3	511	17.22%	6	494	26.32%	6	510	12.35%
50	3	1101	57.51%	3	692	19.72%	5	1029	40.81%	5	740	15.52%

Table 3: Impact of providing an initial solution to the BC as described in Section 4.1 in combination with the custom branching proposed in Section 4.3.

assessment, we select AJI++ IS+Br as the baseline for the remaining experiments in the paper.

5.5. Analyzing the impact of release dates

Finally, we shift the focus to provide some insights regarding the efficiency of the algorithms as well as for the structure of the optimal solutions in terms of the characteristics of the instances. By construction, the release dates depend on the parameter β. Figure 3 illustrates the behavior for the completion time of the optimal solutions and the computation time required by AJI++ IS+Br for the instances with $n = 20$ as a function of β and plotted by instance type (see Section 5.1). As expected, the completion time increases as β increases, meaning that the more spread out release dates are (i.e., higher value of β), the later the distribution will be completed. In particular, clustered instances have the lower completion time. This is consistent with the behaviour we observe on the number of routes. Although not explicitly reported, on average, 2 additional routes are required in the optimal solution per additional unit of β. Therefore, as β increases, there is a preference for visiting customers as soon as they become available. We also observe that the time required by the BC increases as β increases. A similar behaviour is reported by Archetti et al. [4].

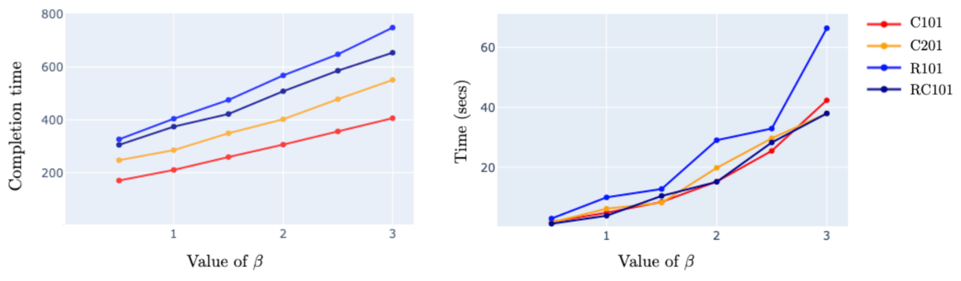


Figure 3: Completion time (left) and BC time in seconds (right) w.r.t. the value of parameter β in instances with $n = 20$ solved by AJI++ IS+Br.

6. TSP-rd variants

Given that release dates are relatively new in the literature, it is interesting to incorporate them and study their impact from an algorithmic perspective into other problem variants. The formulation AJI++ from Section 3.2 can be adapted to other variants of the TSP-rd considering different objective functions and operational constraints. In this section we explore such adaptations. In all cases, an adapted version of the Time-Explorer heuristic described in Section 4.1 and the custom branching scheme proposed in Section 4.3 are used.

6.1. TSP-rd(distance)

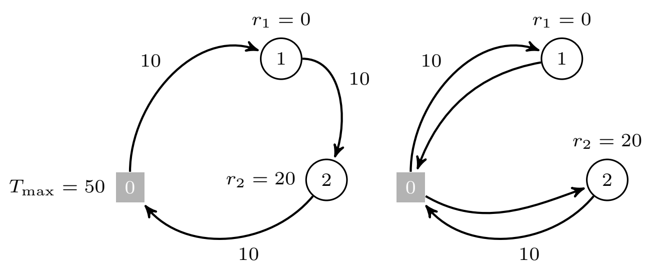
Archetti et al. [5] introduce the TSP-rd(distance) as a variant of the TSP-rd(time) in which an upper bound, called $T_{\max}$, is imposed to the completion time and the objective is to minimize the total traveled distance. Then, in AJI++ the objective (17) is replaced by:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{31}$$

where c_{ij} denotes the distance incurred when traveling the edge (i, j). Note that the distance c_{ij} and the travel time t_{ij} do not need to be the same. However, in Archetti et al. [5] the variant TSP-rd(distance) is introduced assuming $c_{ij} = t_{ij}$ for the sake of simplicity. In addition, the following constraint is added to model the deadline $T_{\max}$ on the completion time:

$$t_{|K|+1} \leq T_{\max} \tag{32}$$

To illustrate how the TSP-rd(time) and TSP-rd(distance) objectives may differ, consider the example depicted in Figure 4 where $c_{ij} = t_{ij}$ for every edge (i, j).



(a) Solution 1
Completion-time: 50
Distance: 30

(b) Solution 2
Completion-time: 40
Distance: 40

Figure 4: Optimal solutions for TSP-rd(distance) (left) and TSP-rd(time) (right), for an instance with two customers $\{1, 2\}$ and $T_{\max} = 50$ for the case of TSP-rd(distance).

Solution (4a) consists of one route which departs at time $20 = \max\{r_1, r_2\}$ and completes the distribution at time 50. The waiting time before departing is 20 and the total traveled distance is 30. Solution (4b) consist of two routes in which the first one departs at time 0, visits Customer 1 ($r_1 = 0$), returns to the depot at time 20, and then departs to visit Customer 2 (whose release date is $r_2 = 20$). The total completion time is 40 as there is no waiting time, and the traveled distance is also 40. Solution (4a) reduces the traveled distance at the expense of a higher completion time, while Solution (4b) increases the traveled distance but reduces the completion time. This highlights how the different objective functions (i.e., completion-time minimization and traveled-distance minimization) may shape the solutions. Note that minimizing completion time, both in AFMG and AJI++, does not guarantee the minimization of traveled distance.

In order to evaluate the adapted AJI++ formulation for the TSP-rd(distance), the instances presented in Section 5.1 require the additional definition of $T_{\max}$, i.e., a new constraint for the distribution that imposes a deadline for the total completion time. In this regard, we introduce the *interval of interesting deadlines* for the TSP-rd(distance), depicted in Figure 5. Let l_1 denote the completion time of the optimal solution for the TSP-rd(time), $r_{\max}$ the latest release date, and z_{TSP} the traveled distance of the optimal TSP solution when discarding the release dates.

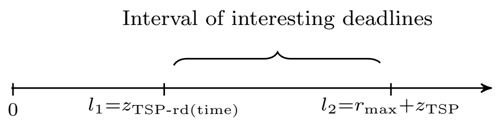

Interval of interesting deadlines
0
$l_1 = z_{\text{TSP-rd(time)}}$
$l_2 = r_{\max} + z_{\text{TSP}}$

Figure 5: *Interval of interesting deadlines* for the TSP-rd(distance).

Proposition 3. *Let $\mathcal{I}$ be an instance of the TSP-rd(distance). If $T_{max} < l_1$, then $\mathcal{I}$ is infeasible.*

Proof. Assume $T_{\max} < l_1$ and let x be a feasible solution for the associated TSP-rd(distance) with completion-time $z(x)$. If the underlying graph G is considered, then x is also a feasible solution for the TSP-rd(time) in G with completion-time $z(x) < T_{\max} < l_1$, which is a contradiction. $\square$

Values of $T_{\max} > r_{\max} + z_{\text{TSP}}$ will not be of interest. The deadline becomes unrestrictive as the solution minimizing travel distance can be obtained by solving the underlying TSP instance (without release dates) as the triangle inequality holds.

For every TSP-rd(time) instance, the start and ending of the quintiles of the interval of interesting deadlines were selected as $T_{\max}$. The greatest value for which the optimal solution of the associated TSP-rd(time) is known is $n = 30$ and, as a result, instances are generated for each $n \in \{20, 25, 30\}$, resulting in a total of 360 instances. Similarly to Archetti et al.[4], for the sake of simplicity, in all instances we set $c_{ij} = t_{ij}$ for all $(i, j) \in A$.

The adapted version of AJI++ to TSP-rd(distance) includes the straightforward adaptation of Algorithm 1 used to obtain an initial feasible solution, in which instances having total completion time greater than $T_{\max}$ are discarded.

Table 4 reports the computational results obtained on the aforementioned TSP-rd(distance) instances. For this experiment, we further indicate in column $T_{\max}$ QU the quintile of the interval of interesting deadlines.

The main insight of Table 4 is that the time required to solve instances decreases as $T_{\max}$ increases. Intuitively, it makes sense as the problem becomes closer to a pure TSP. Moreover, the problem suffers from allowing multiple solutions with the same traveled distance but different completion time as depicted in Figure 6. As $T_{\max}$ decreases, the number of such solutions decreases, arguably making less likely to encounter feasible primal solutions during the enumeration of the BC tree. In this regard, we observe that the computation time required to find the first feasible solution during the BC is larger for small values of $T_{\max}$. In addition, we observe that the adapted version of Algorithm 1 only finds feasible solutions for 60% of the instances, and in particular, only in 2 out of 72 instances with $T_{\max}$ QU= 1 (i.e., $T_{\max} = l_1$). It would be interesting to explore more sophisticated methods to compute initial solutions, aiming to reduce the time required to solve instances with tight values of $T_{\max}$.

Regarding the number of instances solved, only 2 instances out of the 360 considered are not solved in less than 3600 seconds (see Table 4, instances with $n = 30$ and $T_{\max}$ QU= 1). Neither Algorithm 1 nor the BC were able to find a feasible solution in these cases. However, if the time limit is relaxed, the method solves the remaining 2 instances in 4151 and 5578 seconds, respectively.

n	$T_{\max}$ QU	Solved	Time
20	1	24	5
	2	24	2
	3	24	1
	4	24	1
	5	24	1
25	1	24	55
	2	24	8
	3	24	4
	4	24	3
	5	24	2
30	1	22	462
	2	24	41
	3	24	13
	4	24	8
	5	24	3

Table 4: Computational results on TSP-rd(distance) instances.

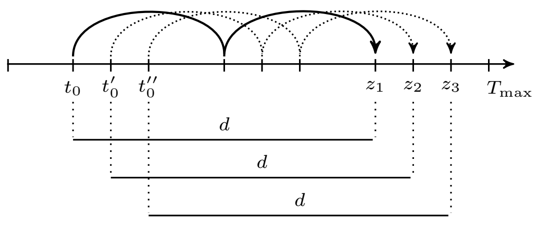


Figure 6: In the context of the TSP-rd(distance), multiple solutions with the same traveled distance d (but different completion-time z_1, z_2 and z_3), may be obtained by shifting the departure t_0 to t'_0 and t''_0 respectively.

6.2. Capacitated TSP-rd(time)

Another natural extension considers a single *capacitated* vehicle, imposing a limit on the total demand to be delivered in each route. Let $C_{\max}$ be the capacity of the vehicle. The following constraints are added to AJI++ formulation from Section 3.2 to model the limit imposed by the capacity:

$$\sum_{i\in N} q_i\, y_i^k \leq C_{\max}\, y_0^k \qquad \forall\, k \in K \tag{33}$$

The value q_i is a one-dimensional representation of the good (e.g., the weight) to be delivered to customer $i \in N$. It is assumed w.l.o.g. that $q_i \leq C_{\max}\ \forall\ i \in N$. The variable y_0^k on the right-hand-side is not actually needed, but improves the linear relaxation of the model.

Remark 4. *Let $\mathcal{I}$ be an instance of the Capacitated TSP-rd(time). Then, Property 1 holds.*

Remark 4 is deduced from the forward shift depicted in Figure 2 which is also feasible for the Capacitated TSP-rd(time). Therefore, constraints (22) are valid as well.

Remark 5. *Let $\mathcal{I}$ be an instance of the Capacitated TSP-rd(time). Then, Property 2 does not hold.*

Remark 5 can be easily proved by considering an instance where more than one route is needed after $r_{\max}$ due to the capacity of the vehicle. As a consequence, constraints (24) must be removed for the TSP-rd(distance).

Moreover, it is possible to tighten the formulation with the following constraint:

$$y_0^{|K|+1-\left\lceil \frac{\sum_{i\in N} q_i}{C_{\max}} \right\rceil} = 1 \tag{34}$$

The rationale is to explicitly provide a lower bound on the number of routes that the vehicle needs to perform.

Instances for the Capacitated TSP-rd(time) are obtained by setting $C_{\max} =$ 100 and assigning to each customer a uniformly random integer q_i in [1, $C_{\max}$]

for every TSP-rd(time) instance with $n \in \{15, 20, 25, 30\}$. The adapted version of AJI++ is run over all the instances and an initial solution is computed with an extension of Algorithm 1 that checks the remaining capacity of the vehicle before adding every customer for routing. It is guaranteed that an initial feasible solution is found.

Table 5 reports columns Solved, Time and GAP as in Table 2. In addition to each value of n, the instance type is reported to highlight the trend observed for the Capacitated TSP-rd(time). The main message is that the problem is more challenging than the uncapacitated version, i.e. the TSP-rd(time). The number of instances solved decreases from 24 to 19 for $n = 20$ and from 24 to 4 for $n = 25$. Moreover, for $n = 30$ no instance is solved to proven optimality within the time limit, and the final average GAP is about 12.95%. The number of routes also increased by 2.55 times w.r.t. the uncapacitated version. Regarding the instance type, it is observed that the number of unsolved clustered instances (i.e., C101 and C201) is bigger than non-clustered instances for $n \in \{20, 25\}$. For $n = 30$, the average GAP is also higher on instances of type C101 and C201. This suggests that the more clustered the customers, the more challenging the TSP-rd(time) instances become when a vehicle capacity and random weights are incorporated. We remark that the model does not incorporate constraints that link both the vehicle capacity and the weights of goods with the release dates, and it would be interesting to study valid inequalities that exploit this connection to enhance the formulation.

n	Instance	Solved	Time	GAP
	C101	6	6	-
	C201	6	9	-
15	R101	6	6	-
	RC101	6	8	-
	C101	3	233	1.93%
	C201	5	1009	1.90%
20	R101	5	278	2.48%
	RC101	6	460	-
	C101	0	-	6.61%
	C201	0	-	7.62%
25	R101	2	1670	4.21%
	RC101	2	1378	6.69%
	C101	0	-	14.87%
	C201	0	-	14.31%
30	R101	0	-	16.08%
	RC101	0	-	6.55%

Table 5: Computational results on Capacitated TSP-rd(time) instances.

6.3. Capacitated TSP-rd(distance)

This version combines the Capacitated TSP-rd and the TSP-rd(distance). Instances require the addition of the value $T_{\max}$ to impose a deadline for the total completion time. Thus, the interval of interesting deadlines (see Figure 5) is replaced with $l_1 = z_{\text{CTSP-rd(time)}}$, i.e. the completion time of the optimal solution of the Capacitated TSP-rd(time), and $l_2 = r_{\max} + z_{\text{CTSP}}$ which corresponds to the value of the optimal solution of the Capacitated TSP where the vehicle departs at $r_{\max}$ and may require to perform multiple routes. A total of 120 instances are generated from each Capacitated TSP-rd(time) instance for $n \in \{10, 15, 20\}$ where the optimal Capacitated TSP-rd(time) solution is known[5].

Table 6 shows the results similarly to Table 4, including also the average final GAP for instances not solved within the time limit. We note that for $n = 10$ all instances are solved in 1 second on average, while for $n = 15$ instances require 15 seconds on average. Several instances remain unsolved for $n = 20$, where only 64 out of 120 instances are solved to proven optimality. Regarding execution time, for $n = 20$ it takes on average 569 seconds, which is significantly larger in comparison with the uncapactitated version, in which all instances are solved in 2 seconds on average. This suggest, once again, that incorporating the vehicle capacity together with random weights make the problem more challenging than the TSP-rd(distance). Finally, it is worth mentioning that, unlike the corresponding uncapacitated variant, the trend regarding the different values of $T_{\max}$ is weaker, but still noticeable.

n	$T_{\max}$ QU	Solved	Time	GAP
10	1	24	0	-
	2	24	0	-
	3	24	1	-
	4	24	1	-
	5	24	1	-
15	1	24	10	-
	2	24	11	-
	3	24	16	-
	4	24	25	-
	5	24	24	-
20	1	13	492	4.61%
	2	14	412	4.42%
	3	13	578	4.67%
	4	12	547	4.55%
	5	12	817	4.81%

Table 6: Computational results on Capacitated TSP-rd(distance) instances.

6.4. *Prize-Collecting TSP-rd(time)*

In this section we introduce the Prize-Collecting variant of the TSP-rd(time). We adapt the definition stated in Vansteenwegen et al. [19] to define the Prize-Collecting TSP-rd(time) where the objective is to find a set of routes performed by a single uncapacitated vehicle that minimizes the total completion time, with the constraint that the total collected profit is at least ρ_{min}. Each customer $i \in N$ contributes with a profit $\rho_i \geq 0$, and it is assumed that ρ_{min} can be collected by visiting all customers in the worst case, i.e., $\sum_{i \in N} \rho_i \geq \rho_{\min}$. However, not all customers need to be visited, but still the constraints imposed by the release

[5]All instances with $n = 20$ are solved to optimality by running the adapted AJI++ with a time limit of 12 hours.

dates have to be satisfied. A first adaptation of the AJI++ formulation to the Prize-Collecting TSP-rd(time) involves the constraints (18), that are replaced by

$$\sum_{k \in K} y_i^k \leq 1 \quad \forall\, i \in N \tag{35}$$

The following result shows that it is not necessary to include such constraints.

Proposition 6. *Constraints (30) and (26) imply constraints (35).*

Proof. Given $i \in N$, consider constraint (30) for $k = |K|$ and note that the left hand side is the same as in (35). Thus, if the left hand side is 0, the inequality is trivially satisfied. Otherwise, vertex i is visited by trip $k \in K$ and the right hand side of (30) becomes 1, thus obtaining (35). $\square$

The following inequality must be further incorporated to account for the total profit collected:

$$\sum_{k \in K} \sum_{i \in N} \rho_i \, y_i^k \geq \rho_{min} \tag{36}$$

As regards the instances, we incorporate profits by following Rule 3 in Bérubé et al. [9]:

$$\rho_i = 1 + \left\lfloor 99 \, \frac{t_{0i}}{\theta} \right\rfloor$$

where $\theta = \max_{j \in N} t_{0j}$. The rule generates hard instances where larger profits are associated with customers that are far from the depot. The parameter ρ_{min} is defined as $\rho_{min} = \alpha \sum_{i \in N} \rho_i$, with $\alpha \in \{0.25, 0.50, 0.75, 0.90\}$, and instances are generated for $n \in \{20,\ 25,\ 30\}$, resulting in a total of 288 instances.

We also adapted Algorithm 1 so that no additional customers are considered once the collected profit is greater than ρ_{min}. Given that $\sum_{i \in N} \rho_i \geq \rho_{\min}$, an initial feasible solution is guaranteed.

Table 7 reports columns Solved, Time and GAP as in Table 2. Results are grouped by n and α. For $n = 20$ all instances are solved to proven optimality in 12 seconds on average. For $n = 25$, only two instances cannot be solved within the time limit, having a final GAP of 1.87% and 1.45% respectively. For $n = 30$ the number of solved instances is 80 out of 96, with an average GAP of 5.49%. Regarding execution time of the BC algorithm, it can be noted that it increases as α increases. This is reasonable given that the higher the value of parameter α, the more customers the vehicle needs to visit to satisfy the minimum profit imposed by ρ_{min}. Therefore, as α increases, the problem becomes closer to the TSP-rd(time).

6.5. Prize-Collecting TSP-rd(distance)

Finally, we consider the variant that minimizes the total traveled distance, formulated as an adaptation of the Prize-Collecting TSP-rd(time) proposed in

n	α	Solved	Time	GAP
20	0.25	24	1	-
	0.50	24	4	-
	0.75	24	22	-
	0.90	24	21	-
25	0.25	24	3	-
	0.50	24	18	-
	0.75	24	290	-
	0.90	22	432	1.66%
30	0.25	24	9	-
	0.50	24	82	-
	0.75	17	548	5.31%
	0.90	15	770	5.66%

Table 7: Computational results on Prize-Collecting TSP-rd(time) instances.

Section 6.4. The objective function is defined by equation (31), and constraint (32) is incorporated to impose a deadline for the completion time. Instances can be generated in a similar fashion by replacing the interval of interesting deadlines from Figure 5 with l_1 being the completion time of the optimal solution for the Prize-Collecting TSP-rd(time), and l_2 corresponding to the value of the optimal solution of the Prize-Collecting TSP departing at $r_{\max}$, both using the same underlying network. Once again, the motivation is to evaluate interesting values of $T_{\max}$, where values of $T_{\max} < l_1$ will result in infeasible instances for the Prize-Collecting TSP-rd(distance), and values of $T_{\max} \geq l_2$ induce instances where release dates are not relevant. Instances are generated for $n \in \{20, 25, 30\}$[6].

Table 8 reports the average execution time required by the BC algorithm over the solved instances, grouped by n, α and the quintiles of $T_{\max}$. Although it is not explicitly reported, only 5 instances (all with $n = 30$ and $T_{\max}$ QU= 1) out of 480 were not solved before the time limit. The main insight of the table is that the required time increases as α increases and $T_{\max}$ decreases. This is consistent with what is observed for the TSP-rd(distance), i.e., the lower the $T_{\max}$, the harder the problem instance, and in variant Prize-Collecting TSP-rd(time), where higher values of α result in more time required to solve the problem instance. Regarding the unsolved instances, in 3 out of 5 the BC algorithm is not able to provide a primal feasible solution within the time limit.

7. Conclusions and future research

In this paper, we propose an alternative formulation for the Traveling Salesman Problem with release dates and completion time minimization, which we

[6]The BC considers an extended time limit of 12 hours in order to solve the Prize-Collecting TSP-rd(time), and only 4 instances with $n = 30$ remained unsolved with an average GAP of 1.24%. The best objective found so far is used to generate Prize-Collecting TSP-rd(distance) instances.

n	α	$T_{\max}$ QU				
		1	2	3	4	5
20	0.25	0	0	1	1	1
	0.50	2	1	1	1	1
	0.75	7	3	2	1	1
	0.90	6	3	2	2	1
25	0.25	2	2	2	2	3
	0.50	10	4	3	3	3
	0.75	181	7	5	3	3
	0.90	208	16	7	5	2
30	0.25	4	4	6	7	6
	0.50	62	12	9	9	6
	0.75	2151	28	16	10	5
	0.90	2070	67	16	11	5

Table 8: Time (in seconds) required to solve Prize-Collecting TSP-rd(distance) instances segmented by parameters α and $T_{\max}$.

use to develop an exact algorithm following a branch and cut scheme. The algorithm is able to solve to optimality instances with up to 30 customers within one hour, outperforming current benchmark from the literature studied by Archetti et al. [4]. An extended set of instances is proposed and our model proved to be able to solve several instances up to 50 nodes. We further extend our formulation to account for other relevant variants of the TSP-rd considering a capacitated vehicle, profits to be collected if a customer is visited and the minimization of the total traveled distance as an alternative objective function to the completion time. We explore variants, analyze some properties and report extensive computational results highlighting the most relevant trends for each case. The objective is not only to find methodological and algorithmic improvements but to also understand the difficulty of each variant. These models may be of value for practitioners seeking for a flexible exact algorithm for Traveling Salesman Problems with release dates. We also aim at opening the discussion around benchmark instances and ILP formulations, by releasing the source code to foster research on these problems.

Several research directions are worth considering as next steps. It would be interesting to perform an in-depth analysis for each variant, as well as strengthening each model with domain specific cuts. Other variants can also be explored, for instance the Profitable TSP-rd should be straightforward to obtain from the AJI formulation. More complex frameworks may be explored, such as Paradiso et al. [14] for the TSP-rd, which would be interesting to evaluate the feasibility of adapting it to the TSP-rd(time), as the time windows must be discarded and the objective function should be modified to account for the completion time. A comparison around the trade-off between performance, flexibility and simplicity of the implementation of such approaches would be of practical interest. Another interesting line of research would be to explore models in which the objective is to minimize distance over the set of solutions with minimum completion-time and it could be interesting to explore bi-level programming models [12] in this regard.

Acknowledgments

The authors are grateful to the two anonymous referees and the Editor in Chief for their valuable and insightful suggestions for improving previous versions of this paper. Additionally, the authors are grateful to Andrea Mor for sharing all the details regarding the instances used in Archetti et al. [4] which helped benchmarking the algorithms. This research has been founded by FONCyT grants PICT-2016-2677 and PICT-2018-2961 from the Ministry of Science, Argentina.

References

[1] Cpu benchmark. `https://www.cpubenchmark.net/compare/Intel-Xeon-E5-1650-v2-vs-Intel-i7-8700/2066vs3099`.

[2] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde. `http://www.math.uwaterloo.ca/tsp/concorde/index.html`.

[3] C. Archetti and L. Bertazzi. Recent challenges in routing and inventory routing: E-commerce and last-mile delivery. *Networks*, 77(2):255–268, 2021.

[4] C. Archetti, D. Feillet, A. Mor, and M. G. Speranza. An iterated local search for the traveling salesman problem with release dates and completion time minimization. *Computers & Operations Research*, 98:24–37, 2018.

[5] C. Archetti, D. Feillet, and M. G. Speranza. Complexity of routing problems with release dates. *European Journal of Operational Research*, 247(3):797–803, 2015.

[6] N. Azi, M. Gendreau, and J.-Y. Potvin. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research*, 178(3):755–766, 2007.

[7] N. Azi, M. Gendreau, and J.-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763, 2010.

[8] N. Azi, M. Gendreau, and J.-Y. Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

[9] J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. A branch-and-cut algorithm for the undirected prize collecting traveling salesman problem. *Networks*, 54(1):56–67, 2009.

[10] D. Cattaruzza, N. Absi, and D. Feillet. The multi-trip vehicle routing problem with time windows and release dates. *Transportation Science*, 50(2):676–693, 2016.

[11] B. Gavish and S. C. Graves. The travelling salesman problem and related problems. Massachusetts Institute of Technology, Operations Research Center, 1978.

[12] T. Kleinert, M. Labbé, I. Ljubić, and M. Schmidt. A Survey on Mixed-Integer Programming Techniques in Bilevel Optimization. working paper or preprint, Jan. 2021.

[13] A. Mor and M. Speranza. Vehicle routing problems over time: a survey. *4OR*, pages 1–21, 2020.

[14] R. Paradiso, R. Roberti, D. Laganá, and W. Dullaert. An exact solution framework for multitrip vehicle-routing problems with time windows. *Operations Research*, 68(1):180–198, 2020.

[15] D. Reyes, A. L. Erera, and M. W. Savelsbergh. Complexity of routing problems with release dates and deadlines. *European Journal of Operational Research*, 266(1):29–34, 2018.

[16] B. C. Shelbourne, M. Battarra, and C. N. Potts. The vehicle routing problem with release and due dates. *INFORMS Journal on Computing*, 29(4):705–723, 2017.

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

[18] L. Taccari. Integer programming formulations for the elementary shortest path problem. *Eur. J. Oper. Res.*, 252(1):122–130, 2016.

[19] P. Vansteenwegen and A. Gunawan. *Orienteering Problems - Models and Algorithms for Vehicle Routing Problems with Profits.* Springer Nature Switzerland AG; Switzerland, 2019.